Adaptive, sample-specific parameter selection for more accurate transcript assembly

Yihang Shen, Zhiwen Yan, and Carl Kingsford*

Ray and Stephanie Lane Computational Biology Department, School of Computer Science,

Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA

Abstract

Motivation: Transcript assemblers are tools to reconstruct expressed transcripts from RNA-seq data. These tools have a large number of tunable parameters, and accurate transcript assembly requires setting them suitably. Because of the heterogeneity of different RNA-seq samples, a single default setting or a small fixed set of parameter candidates can only support the good performance of transcript assembly on average, but are often suboptimal for many individual samples. Manually tuning parameters for each sample is extremely time consuming and requires specialized experience. Therefore, developing an automated system that can advise good parameter settings for individual samples becomes an important problem.

Results: Using Bayesian optimization and contrastive learning, we develop a new automated parameter advising system for transcript assembly that can generate sets of sample-specific parameter candidates. Our framework achieves efficient sample-specific parameter advising by learning parameter knowledge from a large representative set of existing RNA-seq samples and transferring the knowledge to unseen samples. We use Scallop and StringTie, two well-known transcript assemblers, to test our framework on two collections of RNA-seq samples. Results show that our new parameter advising system significantly outperforms the previous advising method in each dataset and each transcript assembler. The source code to reproduce the results from this study can be found at https://github.com/Kingsford-Group/autoparadvisor.

*To whom correspondence should be addressed: `carlk@cs.cmu.edu`

1 Introduction

One crucial computational problem in RNA-seq analysis is transcript assembly—the reconstruction of full-length expressed transcripts from the reads in a given RNA-seq sample. This problem is challenging due to the diversity of splice variants and biases from the technology such as unevenness of read coverage [Shao and Kingsford, 2017]. Although a large number of algorithms have been proposed for assembling different types of RNA-seq samples [Shao and Kingsford, 2017, Zhang et al., 2021, Trapnell et al., 2010, Guttman et al., 2010, Tomescu et al., 2013, Song et al., 2016, Liu et al., 2016, Pertea et al., 2015, Mao et al., 2020, Kovaka et al., 2019, Tung et al., 2019, Liu et al., 2019, Nip et al., 2020], according to benchmarking studies [Voshall and Moriyama, 2018] and the performance reported in recent assemblers, the accuracy of these transcript assemblers is still low. Hence, new strategies are needed for improving the performance of transcript assembly.

Transcript assemblers usually have a large number of tunable parameters. Previous work [DeBlasio et al., 2020, Kovaka et al., 2019] shows that the choices of these parameters significantly affect performance. A single default setting of these parameters, although carefully tuned by the algorithm designers, is typically not optimal for all samples. Manually tuning parameters for a specific sample often leads to higher performance, but it is time consuming. Therefore, developing an automatic parameter advising system (or parameter advisor), a system that can automatically provide good parameter choices for each sample, is an important computational problem. The solution to this problem will provide an orthogonal strategy for improving the accuracy of transcript assembly: rather than developing new transcript assemblers with new algorithmic approaches, we further improve the performance of existing methods by choosing better parameters.

Previous studies developed the parameter advising system by deriving a fixed set of good parameters from data, commonly referred to as an advisor set. However, a key limitation of employing a fixed advisor set is that the fixed set can only guarantee good performance on average [Balcan et al., 2021], but is often suboptimal for many individual samples. This issue becomes more pronounced with smaller advisor sets, which are common in practical applications. A smaller set, while reducing computational load, often fails to capture the variability across different samples, leading to suboptimal performance. Conversely, larger advisor sets significantly prolong the running time of the parameter advisor, as it must evaluate each parameter in the set to identify the most suitable one. Thus, there is a need for a method that can create a compact, sample-specific advisor set. Such an approach would not only enhance accuracy but also streamline the computational process involved in parameter tuning.

We propose the first adaptive, sample-specific parameter advising system for transcript assembly. Our algorithm is based on the intuition that if two samples are "similar", then the optimal parameters for one should be effective for the other. This system operates on a foundational principle: given a set of RNA-seq samples with known optimal parameters (referred to as the training set) and a metric to gauge sample similarity, the advisor set for any new sample R is the set of the best parameters of

the closest samples in the training set, as determined by our similarity measure.

The construction of a sample-specific advisor set involves two challenging sub-tasks: (1) efficiently determining the best parameters for a finite collection of existing RNA-seq samples, and (2) devising a similarity measure that facilitates the transfer of parameter knowledge to novel samples. Neither of these tasks has been adequately addressed in prior research.

To address the first sub-problem, we develop a Bayesian Optimization framework that can efficiently obtain the best parameters of a large number of representative RNA-seq samples of the universe of all possible RNA-seq samples. For the second sub-problem, we develop a contrastive learning framework. This framework is adept at transferring the parameter knowledge to new samples, thereby enabling the identification of new advisor sets.

We evaluate our system, called Auto-ParAdvisor, on the transcript assemblers Scallop [Shao and Kingsford, 2017] and StringTie [Pertea et al., 2015], and compare it against previous parameter advising work on two RNA-seq datasets. Our results show Auto-ParAdvisor significantly outperforms the previous methods in both datasets and for each of the transcript assemblers.

More accurate transcript assembly via Auto-ParAdvisor will benefit downstream RNA-seq analysis such as transcript quantification and differential expression analysis [Shao and Kingsford, 2017]. Moreover, the framework developed here is general and should apply to other bioinformatics tools.

2 Related work

2.1 Parameter advising

Parameter advising has been previously discussed in several bioinformatic areas such as multiple sequence alignment [Cedillo et al., 2022, DeBlasio and Kececioglu, 2014, 2015a,b], biological pathway reconstruction [Magnano and Gitter, 2021] as well as transcript assembly [DeBlasio et al., 2020]. The method proposed in DeBlasio et al. [2020] is the first and only work developing a parameter advising system for transcript assembly by learning the advisor set in a data-driven way. Although the method achieves significantly better performance than the default setting of transcript assemblers on Scallop [Shao and Kingsford, 2017] and StringTie [Pertea et al., 2015], the method is limited by its reliance on a fixed advisor set for all new samples. This limitation is compounded by the small number of RNA-seq samples used in the training set, which fails to comprehensively represent the diversity of all RNA-seq samples. Consequently, this approach does not work well when encountering new samples markedly different from those in the training set, a common issue with small training datasets.

Our current work addresses these shortcomings by employing a large, representative collection of RNA-seq samples. This allows for the learning of a system capable of generating sample-specific advisor sets, thereby offering a more adaptable and robust solution in the context of transcript assembly.

2.2 Bayesian Optimization

Bayesian Optimization (BO) is a sample-efficient framework designed for globally optimizing a black-box function $f(x)$ that is costly to evaluate [Frazier, 2018, Shahriari et al., 2015]. It aims to obtain the input point that maximizes (or minimizes) the function f by iteratively acquiring input points that are likely to achieve the maximum and evaluating the function on these points. Each iteration of BO has two components: a Bayesian statistical model, usually a Gaussian Process regressor [Schulz et al., 2018], to estimate $f(x)$, and an acquisition function that determines the next input point to sample. We refer the reader to Frazier [2018] for a detailed description of BO. Although BO has been successfully applied in many scenarios such as hyper-parameter tuning [Snoek et al., 2012, Klein et al., 2017], reinforcement learning [Brochu et al., 2010, Marco et al., 2017, Wilson et al., 2014], and chemical design [Griffiths and Hernández-Lobato, 2020, Negoescu et al., 2011], its efficiency in tuning parameters of bioinformatics tools has not been broadly explored.

In our study, we introduce the use of Bayesian Optimization (BO) as an effective method for determining the optimal transcript assembly parameters for a specific RNA-seq sample. We use a transcript assembler paired with a loss function designed to estimate the assembler's performance. In this scenario, the task of finding the best parameters for a given RNA-seq sample R is equivalent to optimizing the loss function. As we demonstrate below, this loss function manifests as a black-box function, hence BO emerges as a highly suitable tool for this optimization process due to its proficiency in handling such black-box scenarios.

2.3 Contrastive Learning

Contrastive Learning (CL) is a class of learning methods that learns by comparing input samples. The goal of CL is to learn embeddings of input samples such that those of "similar" samples (e.g. two cat images) are embedded close together, while those of "dissimilar" samples (e.g. one cat image and one dog image) are placed further away [Le-Khac et al., 2020]. A typical CL framework has three components:

- Similarity definition: this determines whether a pair of samples in the training set is similar (positive pair) or dissimilar (negative pair). The criteria for similarity vary depending on the specific problem. For instance, in supervised learning, the labels of input samples might be used to define similarity: samples sharing a label are considered similar, and those with different labels are dissimilar [Khosla et al., 2020].
- Feature encoder: a trainable neural network that maps each input sample into an embedding space.
- Contrastive loss function: this function, using the outputs of the feature encoder and the defined similarity, guides the training process. By minimizing this loss, the encoder learns to map samples from a positive pair close together and those from a negative pair far apart in the embedding space.

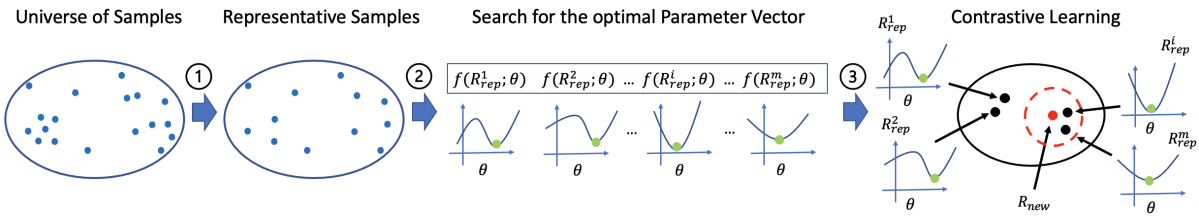

Figure 1: Overview of Auto-ParAdvisor. Step 1: Representative sample selection from $\mathcal{R}$, each blue dot represents one RNA-seq sample, and the blue ellipse represents the universe of all samples; Step 2: Search for the best parameter vectors for representative samples via Bayesian optimization, each green dot represents the optimal point of a function; Step 3: Learning a similarity measure $\mathbb{S}$ via contrastive learning. Samples with similar function shapes are mapped close. The dashed red circle represents the region of the nearest neighbors of a new sample R_{new} under $\mathbb{S}$.

CL has been effectively applied in various domains such as self-supervised learning [Chen et al., 2020, Chuang et al., 2020], image classification [Khosla et al., 2020], gaze direction regression [Wang et al., 2022] and metagenomics [Zhang et al., 2022]. In this work, we adapt CL to compare different RNA-seq samples. We define "similarity" in the context of RNA-seq samples (details below) based on the premise that similar samples will likely have similar optimal parameter vectors for transcript assembly. Thus, knowing the optimal parameters for one RNA-seq sample allows us to apply them to a similar sample, bypassing the need for fresh optimization.

3 Methods

3.1 Overview of Auto-ParAdvisor

Let $\mathcal{R}$ be the universe of all possible RNA-seq samples and $\mathcal{T}$ be the universe of all sets of transcripts. A transcript assembler, denoted as $\mathcal{A}$, is a map $\mathcal{A}(R;\theta) : \mathcal{R}\times\Theta \to \mathcal{T}$ which takes a sample $R \in \mathcal{R}$ and a parameter vector $\theta \in \Theta \subseteq \mathbb{R}^n$ in some parameter space Θ as inputs and outputs a set of transcripts. Given some loss function $\mathcal{L} : \mathcal{T} \to \mathbb{R}$ that estimates the performance of transcript assembly, we use $f(R;\theta) \triangleq \mathcal{L}\circ\mathcal{A}(R;\theta) : \mathcal{R}\times\Theta \to \mathbb{R}$ to estimate the performance of assembler $\mathcal{A}$ with the input sample R and the parameter vector θ. We define $f_R(\theta) \triangleq \mathcal{L}(\mathcal{A}(R;\theta))$ for cleaner notation. The objective in creating a parameter advising system is to develop an oracle $\mathcal{O} : \mathcal{R} \to \Theta$, which can accurately determine an optimal or near-optimal hyper-parameter vector for each RNA-seq sample, that is:

$$f_R(\mathcal{O}(R)) \approx \min_{\theta\in\Theta} f_R(\theta). \tag{1}$$

Our method, Auto-ParAdvisor, constructs such an oracle by first developing a mapping $\mathcal{M}$. This mapping takes an RNA-seq sample R as the input, and outputs an advisor set, a set of good parameter candidates, of this sample. Using this mapping, we then define the oracle $\mathcal{O}_{\mathcal{M}}$ as the

following:

$$\mathcal{O}_{\mathcal{M}}(R) \triangleq \underset{\theta \in \mathcal{M}(R)}{\arg\min}\, f_R(\theta). \tag{2}$$

Different from traditional hyper-parameter tuning in machine learning, which focuses on optimizing hyper-parameters for a single specific function, $\mathcal{M}$ operates as a meta-framework. This framework is able to minimize $f_R(\theta)$ for each sample R by producing sample-specific advisor sets. A significant advantage of $\mathcal{M}$ is that once it is trained, it does not require retraining for new samples, thus offering a more efficient and adaptable approach to hyper-parameter tuning across diverse samples.

As shown in Fig. 1, training Auto-ParAdvisor is composed of three steps: (i) choose a representative sample set, denoted as $\mathcal{R}_{rep}$, from the sample universe $\mathcal{R}$; (ii) search for the best parameter vector for each representative sample $R^i_{rep} \in \mathcal{R}_{rep}$ via Bayesian Optimization; and (iii) develop a similarity measure $\mathbb{S}$ for RNA-seq samples through contrastive learning by using the information from step (ii). The culmination of these steps results in the definition of our mapping $\mathcal{M}$. Specifically, $\mathcal{M}$ is defined as the union of the optimal parameter vectors from those representative samples that are identified as the nearest neighbors of a given sample R under the similarity measure $\mathbb{S}$. That is:

$$\mathcal{M}(R) \triangleq \left\{ \hat{\theta}^i \mid i \in \{1, \ldots, |\mathcal{R}_{rep}|\} \text{ s.t. } R^i_{rep} \in \mathcal{NN}_p\left(R, \mathcal{R}_{rep}, \mathbb{S}\right) \right\}, \tag{3}$$

where $\hat{\theta}^i$ represents the optimal parameter vector for the representative sample R^i_{rep} as determined through Bayesian Optimization in step (ii), and $\mathcal{NN}_p\left(R, \mathcal{R}_{rep}, \mathbb{S}\right)$ is the set of p samples within $\mathcal{R}_{rep}$ that are closest to the sample R under the measure $\mathbb{S}$. We now use the following three sections to describe the three steps (i) to (iii) in detail.

3.2 Representative sample selection from $\mathcal{R}$

To construct $\mathcal{M}$, it is essential to use existing RNA-seq samples as the training foundation. The training set needs to effectively encapsulate the critical transcriptional characteristics of $\mathcal{R}$, the entire spectrum of RNA-seq samples, to ensure that $\mathcal{M}$ possesses strong generalizability to new samples. One approach could be to incorporate all existing RNA-seq samples into the training set. However, considering the immense volume of available RNA-seq data – numbering in the hundreds of thousands – such an approach would demand prohibitively high computational resources. To strike a balance between comprehensive representation and computational feasibility, we adopt the representative sample set identified in Tung and Kingsford [2021], which includes around 7,000 human RNA-seq samples. This set effectively represents the broader collection of 196,523 existing human RNA-seq samples. To enhance the computational efficiency, we further use apricot [Schreiber et al., 2020], for selecting a smaller representative subset from these samples. This selection process ensures a robust coverage of $\mathcal{R}$, while significantly alleviating computational demands. The final subset of representative samples selected through this method is denoted as $\mathcal{R}_{rep}$.

3.3 Search for the best parameter vectors for representative samples

For every representative sample R^i_{rep} in $\mathcal{R}_{rep}$, we use BO to search for its best parameter vector by minimizing the function $f_{R^i_{rep}}(\theta)$. BO is particularly suitable for this task for two main reasons: (i) $f_{R^i_{rep}}(\theta)$ is essentially a black-box function, meaning its derivatives are difficult to compute. This complexity arises because the transcript assembler $\mathcal{A}$ typically comprises multiple modules that tackle intricate combinatorial problems. As a result, it is not feasible to express $f_{R^i_{rep}}(\theta)$ in a closed form; and (ii) each evaluation of $f_{R^i_{rep}}(\theta)$ necessitates a run of the transcript assembler $\mathcal{A}$ which can be a slow process, ranging from minutes to hours. Therefore, even a single evaluation of $f_{R^i_{rep}}(\theta)$ can be quite time-consuming.

However, standard Bayesian Optimization algorithms are not immediately applicable in our context due to their reliance on a predefined search domain. Typically, these algorithms require a user-specified domain, assumed to encompass the optimal or near-optimal points for the function being optimized [Shahriari et al., 2016]. In the case of RNA-seq samples, however, the challenge arises from the variability of optimal parameter vectors across different samples. Given this diversity, and the potential for significant dissimilarities between optimal parameters for various samples, it becomes impractical to establish a fixed search domain suitable for all cases. This aspect of the RNA-seq samples necessitates an adaptation or modification of standard Bayesian Optimization techniques to accommodate the varying nature of the optimal parameter vectors.

Inspired by Poloczek et al. [2016] and DeBlasio et al. [2020], we propose a new warm-up-based BO, called CAWarm-BO (Coordinate Ascent Warm-up Bayesian Optimization), that is able to automatically adjust the search domain for each sample. The effectiveness of this new framework is based on the following observations:

- Coordinate ascent is an iterative optimization algorithm that successively maximizes along the coordinate directions of a function to find the maximum [Zangwill, 1969]. Coordinate ascent does not require a predefined search domain, and it will rapidly converge towards a local region potentially containing the optimal or near-optimal points. However, within that localized region, coordinate ascent tends to be slower in pinpointing the local optimal point.
- Bayesian Optimization is able to efficiently find the best point within a reasonably sized search domain, but the search domain must be predefined.

By synergizing these two methodologies, CAWarm-BO leverages the strengths of both: using coordinate ascent to swiftly approach a promising region without the need for a predefined domain, followed by Bayesian Optimization to efficiently find the optimal point within that localized area.

We run CAWarm-BO independently on each sample in the representative set $\mathcal{R}_{rep}$ (here we maximize $-f_{R^i_{rep}}(\theta)$ which is equivalent to minimizing $f_{R^i_{rep}}(\theta)$). For each function $f_{R^i_{rep}}$, CAWarm-BO yields a set of pairs denoted as $\mathcal{X}^i_{rep} = \left\{\left(\theta^i_g, y^i_g\right)\right\}_{g=1}^{N}$. In this set, θ^i_g represents a queried parameter vector during the Bayesian Optimization search, and $y^i_g = f_{R^i_{rep}}(\theta^i_g)$ corresponds to the loss value for that parameter vector. The optimal parameter vector $\hat{\theta}^i$ for sample R^i_{rep} is determined

as the parameter vector θ_g^i in $\mathcal{X}_{rep}^i$ that yields the smallest y_g^i.

3.4 Learning the similarity measure via contrastive learning

Compared to other optimization algorithms, CAWarm-BO is more efficient in optimizing the function $f_R(\theta)$ for a given sample R. However, as shown in our experimental results it is still not efficient enough that we can directly use it to search for the best parameter vector for every new sample, i.e. it cannot be solely relied upon as the oracle $\mathcal{O}$ due to its slow speed. This limitation underscores the need to develop a similarity measure $\mathbb{S}$, enabling model $\mathcal{M}$ to quickly generate the specific advisor set for each new sample through nearest neighbor search, as shown in Eq. 3. We learn $\mathbb{S}$ via the contrastive learning (CL) framework. Within this framework, given any two RNA-seq samples R_1 and R_2, they are embedded into a latent space via a feature encoder Enc_ϕ with trainable parameters ϕ. In this latent space, the similarity between two vectors is quantified using the cosine similarity measure:

$$\mathbb{S}(R_1, R_2) \triangleq \frac{Enc_\phi(R_1)^\top Enc_\phi(R_2)}{\|Enc_\phi(R_1)\|\|Enc_\phi(R_2)\|}. \tag{4}$$

Learning $\mathbb{S}$ is now equivalent to training the encoder Enc_ϕ. Our training framework is inspired by previous work of supervised contrastive learning such as Khosla et al. [2020] and Wang et al. [2022], but has the following several unique components:

- We represent each sample in $\mathcal{R}_{rep}$ by a set derived from the MinHash sketch [Broder, 1997, Ondov et al., 2016]. These sets serve as the inputs for the encoder Enc_ϕ. The details are described in Section 3.4.1.
- We develop a novel method to define the similarities between pairs of representative samples, denoted as $\hat{\mathbb{S}}_{rep}(R_{rep}^i, R_{rep}^j)$ for training purposes. The details are described in Section 3.4.2.
- We develop a distinctive data augmentation module for enhancing the robustness and accuracy of training. The details are described in Section 3.4.3.
- We develop a neural network architecture and a contrastive loss function that are particularly tailored to our specific requirements. The details are described in Section 3.4.4.

The pseudocode of the training framework is given in Algorithm 1. During each training iteration, every input sample in the training batch is embedded using the encoding network Enc_ϕ (line 5 of Alg. 1). The outputs from Enc_ϕ, together with $\hat{\mathbb{S}}_{rep}$, are used to compute the contrastive loss function ($\mathcal{L}_{CL}$) (line 7 of Alg. 1). We train Enc_ϕ by minimizing $\mathcal{L}_{CL}$ (line 8 of Alg. 1) via gradient-based algorithms.

To obtain an advisor set for a new sample, we use the learned Enc_ϕ to map this new sample to the embedding space (line 3 of Alg. 2) and use the cosine similarity to identify the closest neighbors among the representative samples (lines 6 of Alg. 2). The advisor set is then composed of the best parameter vectors from these nearest neighbors (lines 8 of Alg. 2).

Algorithm 1 Contrastive Training Module

Input Input MinHash sets $\{X^i\}_{i=1}^n$ with size n, similarities between representative samples $\hat{\mathbb{S}}_{rep}$, initialized encoder Enc_ϕ, batch size N_b, total epochs N_{ep}, total number of batches $tot_batch = \left\lceil \frac{n}{N_b} \right\rceil$
for $epoch = 1, \ldots, N_{ep}$ **do**
  for $batch = 1, \ldots, tot_batch$ **do**
    for $i = 1, \ldots, N_b$ **do**
      $z^i_{batch} = Enc_\phi(X^i_{batch})$
    end for
    compute the contrastive loss $\mathcal{L}_{CL}\left(\left\{z^i_{batch}\right\}_{i=1}^{N_b}, \hat{\mathbb{S}}_{rep}\right)$ via Eq. 9
    update ϕ to minimize $\mathcal{L}_{CL}$
  end for
end for
return Enc_ϕ

Algorithm 2 Nearest Neighbor Computation for a New Sample

Input representative samples $\{R^i_{rep}\}_{i=1}^m$ and their corresponding input sets $\{X^i_{rep}\}_{i=1}^m$, new RNA-seq sample R_{new}, trained encoder Enc_ϕ, constant p
MinHash sketch on R_{new} to obtain the new input set X_{new}.
$z_{new} = Enc_\phi(X_{new})$
for $q \in \{1, \ldots, m\}$ **do**
  $z_q = Enc_\phi(X^q_{rep})$
  compute cosine similarity between vectors z_q and z_{new}, $S^q_{cos} \triangleq \frac{z_{new}^\top z_q}{\|z_{new}\|\|z_q\|}$
end for
Sort $\{S^q_{cos}\}_{q=1}^m$ and pick p samples in the representative set that have highest p similarity values with R_{new}.
return p samples

3.4.1 RNA-seq sample representation

Given that each RNA-seq sample comprises a vast number of sequencing reads, which are not ideally suited for direct input into neural networks, we use the MinHash sketch [Broder, 1997, Ondov et al., 2016] to generate representations for these samples. MinHash sketch process begins by decomposing the reads in an RNA-seq sample into constituent k-mers. Each k-mer is then processed through a hash function h to obtain a hash value. Finally, for a predetermined sketch size s, MinHash sketch returns the s smallest hashes output by h over all k-mers, along with the count of k-mers associated with these hash values. Each RNA-seq sample R is represented by a set input $X = \{x_i\}_{i=1}^s$, where each x_i is a two-dimensional vector; the first dimension denotes a hash value, and the second dimension indicates the number of k-mers in R corresponding to this hash.

3.4.2 Similarity between representative samples

Contrary to supervised contrastive learning in classification problems [Khosla et al., 2020] where sample pairs are simply categorized as positive (having the same label) or negative (having different labels), our case more closely aligns with a contrastive regression problem as in Wang et al. [2022] where $\hat{\mathbb{S}}_{rep}$ is used to assess the degree of similarity between two RNA-seq samples. Ideally, for each pair of representative samples (R^i_{rep}, R^j_{rep}), $\hat{\mathbb{S}}_{rep}(R^i_{rep}, R^j_{rep})$ should correspond to the similarity between their respective functions $(f_{R^i_{rep}}, f_{R^j_{rep}})$. However, since these functions lack a closed-form

representation, direct computation of their similarity is not feasible. Instead, we leverage information from the Bayesian Optimization (BO) step to approximate these functions and assess their similarity. Specifically, as described above, the output of CAWarm-BO for each sample R^i_{rep} is a set of pairs $\mathcal{X}^i_{rep} = \left\{\left(\theta^i_g, y^i_g\right)\right\}_{g=1}^N$, hench we use Gaussian Process ($\mathcal{GP}$) models to fit data: $\mathcal{X}^i_{rep} \sim \mathcal{GP}^i_{rep}$ and $\mathcal{X}^j_{rep} \sim \mathcal{GP}^j_{rep}$. The posterior means of the Gaussian Process models, denoted as μ^i_{rep} and μ^j_{rep} respectively, are estimations of functions $f_{R^i_{rep}}$ and $f_{R^j_{rep}}$. We then introduce the following term, called *normrank*, to quantify the similarity between μ^i_{rep} and μ^j_{rep}:

$$normrank\left(R^i_{rep}, R^j_{rep}\right) \triangleq \frac{\left|\left\{\theta^i_g \mid \left(\theta^i_g, y^i_g\right) \in \mathcal{X}^i_{rep} \text{ and } \mu^i_{rep}\left(\theta^i_g\right) < \mu^i_{rep}(\hat{\theta}^j)\right\}\right|}{\left|\mathcal{X}^i_{rep}\right|}, \tag{5}$$

where $\hat{\theta}^j$ is the optimal parameter vector for R^j_{rep} found by BO. The numerator of Eq. (5) represents the number of parameter vectors queried in $\mathcal{X}^i_{rep}$ that have smaller posterior mean values than the value of the optimal parameter vector for R^j_{rep}. Intuitively, if $f_{R^i_{rep}}$ and $f_{R^j_{rep}}$ are similar, the optimal parameter vector for R^j_{rep} should also be an approximately optimal one of R^i_{rep}, making the numerator smaller. The denominator of Eq. 5 serves as a normalization term, accounting for different sizes of $\mathcal{X}^i_{rep}$.

normrank functions as a distance-like measure between samples: the more alike the samples are, the lower the *normrank* value. However, it has two drawbacks: (i) it is not symmetric, meaning $normrank\left(R^i_{rep}, R^j_{rep}\right)$ may not be equal to $normrank\left(R^j_{rep}, R^i_{rep}\right)$, and (ii) it does not constitute a true distance as it fails to satisfy the triangle inequality. To address these issues, we first compute the average $\left(normrank\left(R^i_{rep}, R^j_{rep}\right) + normrank\left(R^j_{rep}, R^i_{rep}\right)\right)/2$ to ensure symmetry. We then apply the shortest-path method, as introduced in Lesne et al. [2014], to reconstruct the distances, denoted as $\hat{\mathbb{D}}_{rep}$, and to filter out some noise. Based on this, $\hat{\mathbb{S}}_{rep}$ is subsequently defined as:

$$\hat{\mathbb{S}}_{rep}\left(R^i_{rep}, R^j_{rep}\right) \triangleq 1 - 2\hat{\mathbb{D}}_{rep}\left(R^i_{rep}, R^j_{rep}\right). \tag{6}$$

According to Eq. (5), for any pair of samples, the value of $\hat{\mathbb{D}}_{rep}\left(R^i_{rep}, R^j_{rep}\right)$ falls within the range $[0, 1]$. Consequently, as per Def. (6), this ensures that the value of $\hat{\mathbb{S}}_{rep}\left(R^i_{rep}, R^j_{rep}\right)$ lies within the range $[-1, 1]$, aligning with the range typical for cosine similarity.

3.4.3 Data augmentation module

The time-intensive nature of conducting Bayesian Optimization (BO) on each RNA-seq sample to acquire the optimal parameter vector imposes a constraint on the size of the representative sample set; it cannot be excessively large. This limitation implies that the amount of training data available for contrastive learning is not vast. As will be discussed in Section 4, the representative sample set comprises around 1,000 samples. To improve the accuracy of training under these constraints, we develop a novel data augmentation method. This method substantially expands the size of the training dataset without necessitating additional rounds of BO computation.

The underlying principle of our data augmentation method is as follows: for a new RNA-seq

sample R_{sub}, created by subsampling reads from an original sample R, its corresponding function $f_{R_{sub}}$ is expected to have a similar optimal parameter vector to that of f_R. This means we can create additional training data by subsampling reads from the original RNA-seq samples, and calculate the similarity scores without the need for extra Bayesian Optimization (BO) computations. Specifically, let R^i_{sub1} and R^j_{sub2} be subsampled versions of R^i_{rep} and R^j_{rep} respectively. The similarity score between these two new subsampled samples is defined as follows:

$$\hat{\mathbb{S}}_{rep}\left(R^i_{sub1}, R^j_{sub2}\right) \triangleq \hat{\mathbb{S}}_{rep}\left(R^i_{rep}, R^j_{rep}\right), \forall i, j \in \{1, 2, \ldots, m\}. \tag{7}$$

In practice, for each representative sample, we generate seven subsampled variants with sampling ratios 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95. This approach effectively increases the size of our training data by a factor of seven, all without necessitating any additional Bayesian Optimization (BO) computations. The effectiveness of this data augmentation module is further discussed in Section 4.4.

3.4.4 Neural network architecture and loss function

As outlined in Section 3.4.1, each RNA-seq sample is represented as a set. Consequently, in our case, it is necessary to use a neural network architecture that is capable of processing sets as inputs. Crucially, this architecture must exhibit permutation invariance, meaning its output should remain unchanged irrespective of the order of elements within the set.

Inspired by Zaheer et al. [2017], the encoder Enc_ϕ is designed to be a composite of two sub-networks: h_φ and g_ψ with trainable parameters φ and ψ. Specifically, given an RNA-seq sample R with its set representation $X = \{x_i\}_{i=1}^s$, we have:

$$Enc_\phi(X) = g_\psi\left(\frac{1}{s}\sum_{i=1}^{s} h_\varphi(x_i)\right). \tag{8}$$

Theorem 2 of Zaheer et al. [2017] proved that the form of Eq. (8) preserves permutation invariance.

Our contrastive loss function is simply defined as mean squared error:

$$\mathcal{L}_{CL}\left(\left\{z^i_{batch}\right\}_{i=1}^{N_b}, \hat{\mathbb{S}}_{rep}\right) \triangleq \frac{1}{N_b^2}\sum_{i=1}^{N_b}\sum_{j=1}^{N_b}\left(\frac{{z^i_{batch}}^\top z^j_{batch}}{\|z^i_{batch}\|\|z^j_{batch}\|} - \hat{\mathbb{S}}_{rep}\left(R^i_{batch}, R^j_{batch}\right)\right)^2, \tag{9}$$

where N_b is the batch size and R^i_{batch} and R^j_{batch} are RNA-seq samples in the training data corresponding to embeddings z^i_{batch} and z^i_{batch}. Compared to the contrastive loss formulas used in Khosla et al. [2020] and Wang et al. [2022], we find that this simple loss function achieves very good performance in our case, with the advantage that no extra hyper-parameters such as the temperature need to be tuned. We leave to future work the investigation of more sophisticated loss functions.

4 Experimental results

4.1 Implementation Details

We test Auto-ParAdvisor on Scallop v0.10.2 ($\mathcal{A}$ = Scallop) with 18 tunable parameters and StringTie v2.2.1 ($\mathcal{A}$ = StringTie) with 8 tunable parameters. The details of these parameters are listed in Table 7 and Table 8 in the appendix. Similar to DeBlasio et al. [2020], we use the area under the curve (AUC) of an assembler's prediction of known transcripts to estimate the performance of these two transcript assemblers (true positive: transcripts that are in both $\mathcal{A}$'s output and the known transcript set; false positive: transcripts that are in $\mathcal{A}$'s output but not in the known transcript set; false negative: transcripts that are in the known transcript set but not in $\mathcal{A}$'s output). The transcript annotation file for GRCh38 can be downloaded from https://ftp.ensembl.org/pub/release-108/gtf/homo_sapiens. The loss function $\mathcal{L}$ is defined as the negative AUC value.

Since transcript assemblers including Scallop and StringTie usually have mixed types of parameters (parameters include categorical, integer, and continuous variables), we choose CASMOPOLITAN [Wan et al., 2021], a BO-based algorithm for mixed search spaces, as the basic framework of CAWarm-BO. Other detailed settings of CAWarm-BO are described in Appendix A.1.

From $196,523$ human samples, $7,860$ human RNA-seq samples are selected by Tung and Kingsford [2021] as representative samples. We then use apricot [Schreiber et al., 2020] to further select $1,500$ samples for constructing $\mathcal{R}_{rep}$. These $1,500$ samples are aligned using the aligner STAR [Dobin et al., 2013] to the human reference genome GRCh38. We run CAWarm-BO independently on these samples, and remove samples that have very low AUC values (the samples for which the best AUC value found by BO is less than 3×10^{-4}). This filtration process resulted in 1,263 samples retained for contrastive training for Scallop, and 1,235 samples for StringTie.

We use PyTorch [Paszke et al., 2019] to implement the contrastive learning framework. We use simple fully connected neural networks with ReLU activations to represent h_φ and g_ψ in Eq. (8). We leave to future work the investigation of optimal neural network architectures. Additional details of this framework are described in Appendix A.2.

As in DeBlasio et al. [2020], we use the following three datasets in our experiments:

- ENCODE10, a collection of 10 RNA-seq samples from the ENCODE database [The ENCODE Project Consortium, 2012]. These samples are already aligned in DeBlasio et al. [2020] to the human reference genome GRCh38. DeBlasio et al. [2020] use this dataset to generate the advisor set, while we use it to test the performance of CAWarm-BO.
- ENCODE65, a collection of 65 RNA-seq samples also from ENCODE. None of them is included in the representative sample set (training data). These samples have pre-existing alignments in ENCODE. We use them to test the performance of Auto-ParAdvisor.
- SRA-test, a collection of 1595 RNA-Seq samples from the Sequence Read Archive [Cochrane et al., 2016]. These samples are aligned using the aligner STAR [Dobin et al., 2013] to the

human reference genome GRCh38. We first remove the samples from SRA that are also in the representative sample set (fewer than 10 samples), and then use the remaining samples to test the performance of Auto-ParAdvisor.

4.2 Performance of CAWarm-BO

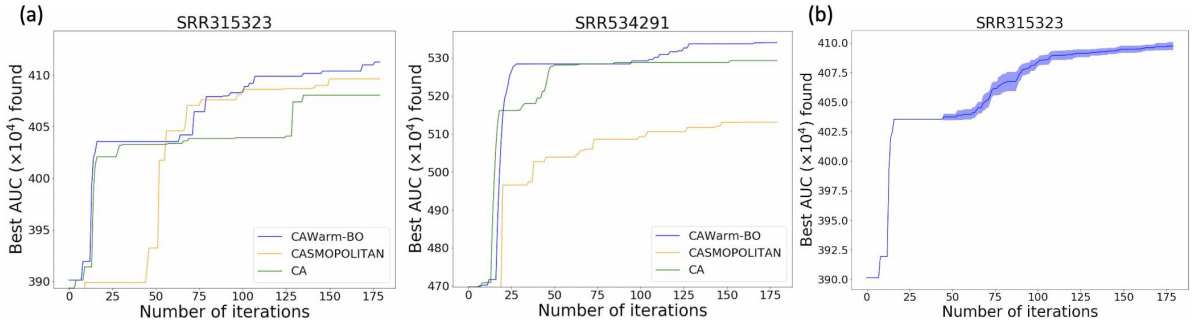

Figure 2: (a) Comparisons of Coordinate Ascent (CA), CASMOPOLITAN and CAWarm-BO on two RNA-seq samples from ENCODE10 (accession numbers: SRR315323 and SRR534291). The transcript assembler used here is Scallop. For each sample, we do one run for each method and plot the maximum value found by iterations. Here ($\times 10^4$) means the AUC values shown in the figures are real AUC values times 10^4. (b) The variance of CAWarm-BO with different runs. Here, we do 10 independent runs on the sample SRR315323, and plot the mean and 1/2 standard deviation of the maximum value found by iterations. Note that the warm-up part is deterministic.

We compare CAWarm-BO with CASMOPOLITAN (without warmup and search domain adjustment) and coordinate ascent (CA) proposed in DeBlasio et al. [2020] on the ENCODE10 dataset. Fig. 2(a) and Fig. 9(a,b,c) show that in most samples, CAWarm-BO's performance closely mirrors CA during the warmup phase but surpasses CA following the commencement of BO. CASMOPOLITAN shows competitive results in many samples when compared with CAWarm-BO, but without a warmup phase, it initially spends several iterations identifying a suitable local search region, resulting in slower convergence within the first 50 iterations. Furthermore, lacking search domain adjustment, CASMOPOLITAN sometimes only achieves a sub-optimal point, as seen in the middle panel of Fig. 2. We also evaluated the highest AUC values obtained by CAWarm-BO and CASMOPOLITAN within 200 iterations against the best values achieved by CA without any iteration limit (continuing CA until the stopping criteria set by DeBlasio et al. [2020] are met). Table 9 in the appendix reveals that the best AUC values discovered by CAWarm-BO and CA are quite similar, although CA requires significantly more iterations than CAWarm-BO to reach these optimal values. Additionally, Table 10 indicates that this trend is consistent across different transcript assemblers. These findings indicate that CAWarm-BO is a more efficient method than both CA and CASMOPOLITAN for finding the best parameter vector for each sample in the representative sample set.

BO is a method that contains randomness, which can lead to variations in the results from different runs of BO on the same function. To test whether this variance significantly impacts the final performance, we executed 10 distinct runs of CAWarm-BO on samples SRR315323 and SRR307903, and plot the mean and 1/2 standard deviation of the maximum value found by iterations.

Fig. 2(b) and Fig. 9(d) demonstrate that the variance among these 10 runs for each iteration is minimal, and the best values obtained from different runs are similar. This consistency indicates that running CAWarm-BO just once per sample is sufficient, and the comparisons shown in Fig. 2(a) and Fig. 9(a,b,c) are reliable and robust.

4.3 Accuracy of similarity definition between representative samples

We use the following correlation test to verify the accuracy of similarity values $\hat{\mathbb{S}}_{rep}$ defined in Section 3.4.2 between representative samples. We randomly select 10 samples in the representative set. For each of these samples, denoted R, we compute $f_R(\theta)$ using the best parameter vectors $\hat{\theta}^i$ found by CAWarm-BO for all other samples in the representative set. We plot the relationships between these function values $\{f_R(\hat{\theta}^i)\}_{i=1}^m$ and the similarity values $\{\hat{\mathbb{S}}_{rep}(R, R_{rep}^i)\}_{i=1}^m$, shown in Fig. 10(c). We compare $\hat{\mathbb{S}}_{rep}$ with two alternative similarity definitions: (i) Mash distance [Ondov et al., 2016], a distance metric based on the set representations of two samples. We calculate this distance between sample R and all others, plotting the function values against the Mash similarity (negative Mash distance values), shown in Fig. 10(a); (ii) The Euclidean distance between the best parameter vector of R (found by CAWarm-BO) and those of all other samples, normalized to the [0,1] range to negate scale effects. The relationship between the function values and the Euclidean similarity (negative Euclidean distance values) is also plotted, shown in Fig. 10(b). An ideal similarity measure should exhibit a strong negative correlation with these function values. Table 1 and Table 2, along with Fig. 10 show that in most tested samples $\hat{\mathbb{S}}_{rep}$ demonstrates a higher negative correlation with the function values compared to both Mash and Euclidean similarities, irrespective of the transcript assembler used. These results suggest that leveraging information from the BO steps enables more accurate quantification of sample similarity than directly computing the distance between set representations of RNA-seq samples. Moreover, the *normrank* function used in $\hat{\mathbb{S}}_{rep}$ incorporates broader information from the BO step than just the best parameter vector of each sample, thus enhancing the accuracy of similarity quantification.

Samples	Mash similarity	Euclidean similarity	$\hat{\mathbb{S}}_{rep}$
ERR197922	-0.153	-0.112	**-0.502**
SRR1812365	-0.460	-0.277	**-0.600**
SRR5099917	-0.353	0.114	**-0.664**
SRR492054	-0.713	-0.363	**-0.891**
SRR346466	-0.619	-0.682	**-0.842**
SRR6687435	-0.372	-0.512	**-0.754**
SRR1782845	-0.556	-0.480	**-0.757**
SRR8535251	-0.327	-0.238	**-0.435**
SRR3932352	-0.627	-0.665	**-0.894**
ERR2619102	-0.254	0.348	**-0.432**

Table 1: Spearman's rank correlation coefficient between similarity measures and function values for different samples. The transcript assembler used here is Scallop.

Samples	Mash similarity	Euclidean similarity	$\hat{\mathbb{S}}_{rep}$
SRR3385209	-0.715	**-0.724**	**-0.724**
SRR8274574	-0.779	-0.574	**-0.860**
SRR5659428	**-0.789**	-0.060	-0.779
SRR4318442	**-0.685**	-0.039	-0.484
SRR6312630	-0.267	-0.632	**-0.806**
SRR8273981	-0.783	-0.623	**-0.817**
SRR5604831	-0.560	-0.614	**-0.691**
SRR5878078	-0.673	-0.544	**-0.703**
SRR9321872	-0.713	-0.704	**-0.747**
SRR1981008	-0.714	**-0.715**	-0.667

Table 2: Spearman's rank correlation coefficient between similarity measures and function values for different samples. The transcript assembler used here is StringTie.

4.4 Rationality and performance of the data augmentation module

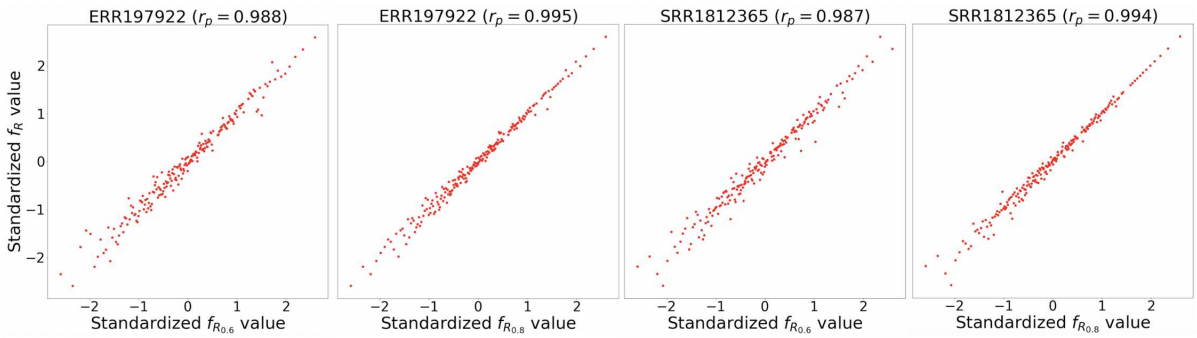

Figure 3: The correlation between two sets of copula standardized [Salinas et al., 2020] function values $\left\{f_{R^i_{0.8}}(\theta^i_g)\right\}_{g=1}^{N}$ ($\left\{f_{R^i_{0.6}}(\theta^i_g)\right\}_{g=1}^{N}$) and $\left\{f_{R^i_{rep}}(\theta^i_g)\right\}_{g=1}^{N}$. Each dot represents one parameter vector θ^i_g. Sample accession numbers: ERR197922 and SRR1812365. The transcript assembler $\mathcal{A}$ used here is Scallop. $R_{0.6}$: sample from the subsampling with ratio 0.6. $R_{0.8}$: sample from the subsampling with ratio 0.8. r_p: Pearson correlation coefficient.

In this section, we first empirically validate the underlying principle of our data augmentation method: a sample R_{sub} created by subsampling reads from an original sample R shares a similar optimal parameter vector with R. We randomly select 10 samples from the representative set. For each selected sample R^i_{rep}, we create a subsampled version $R^i_{0.8}$ with a sampling ratio of 0.8, meaning $R^i_{0.8}$ contains 80% of the reads in R^i_{rep}. We then compute $f_{R^i_{0.8}}(\theta^i_g)$ for each query point θ^i_g in $\mathcal{X}^i_{rep}$ and calculate the correlation between $\left\{f_{R^i_{0.8}}(\theta^i_g)\right\}_{g=1}^{N}$ and $\left\{f_{R^i_{rep}}(\theta^i_g)\right\}_{g=1}^{N}$, where N is the size of $\mathcal{X}^i_{rep}$. This procedure is repeated with a sampling ratio of 0.6. If the functions $f_{R^i_{rep}}$ and $f_{R^i_{0.8}}$ (or $f_{R^i_{0.6}}$) are simialr, we would expect $f_{R^i_{rep}}(\theta^i_g) \approx f_{R^i_{0.8}}(\theta^i_g)$ and $f_{R^i_{rep}}(\theta^i_g) \approx f_{R^i_{0.6}}(\theta^i_g)$ for each θ^i_g. High correlations, as shown in Fig. 3, Table 3, and Table 4, indicate that for both transcript assemblers Scallop and StringTie, when reads in R^i_{rep} are randomly subsampled with an adequate ratio, the resulting function closely resembles the original.

To confirm that the set representation of $R^i_{0.8}$ or $R^i_{0.6}$ differs from R^i_{rep} (ensuring actual data augmentation rather than duplication), we also compute the Mash distance between these two set

representations. The non-zero distances reported in Table 5 and Table 6 confirm that these samples have distinct set representations. Moreover, the values in both tables are considerably lower than the average Mash distances among samples in the representative set (0.0728 for Scallop, and 0.0624 for StringTie). These results show that the set representation of $R^i_{0.8}$ or $R^i_{0.6}$ is distinct from, but similar to, R^i_{rep}, validating the efficacy of read subsampling as a reasonable data augmentation approach.

Samples	Subsample (0.6)	Subsample (0.8)
ERR197922	0.988	0.995
SRR1812365	0.987	0.994
SRR5099917	0.975	0.981
SRR492054	0.935	0.975
SRR346466	0.950	0.977
SRR6687435	0.979	0.992
SRR1782845	0.968	0.990
SRR8535251	0.888	0.937
SRR3932352	0.979	0.991
ERR2619102	0.954	0.976

Table 3: Pearson correlation coefficient between two sets of function values of $f_{R^i_{0.6}}$ ($f_{R^i_{0.8}}$) and $f_{R^i_{rep}}$ in 10 different samples. The transcript assembler used here is Scallop.

Samples	Subsample (0.6)	Subsample (0.8)
SRR3385209	0.920	0.974
SRR8274574	0.968	0.993
SRR5659428	0.983	0.995
SRR4318442	0.728	0.914
SRR6312630	0.828	0.914
SRR8273981	0.915	0.987
SRR5604831	0.946	0.974
SRR5878078	0.961	0.970
SRR9321872	0.785	0.966
SRR1981008	0.911	0.829

Table 4: Pearson correlation coefficient between two sets of function values of $f_{R^i_{0.6}}$ ($f_{R^i_{0.8}}$) and $f_{R^i_{rep}}$ in 10 different samples. The transcript assembler used here is StringTie.

Samples	Subsample (0.6)	Subsample (0.8)
ERR197922	0.0080	0.0033
SRR1812365	0.0083	0.0034
SRR5099917	0.0071	0.0031
SRR492054	0.0218	0.0099
SRR346466	0.0111	0.0040
SRR6687435	0.0087	0.0033
SRR1782845	0.0145	0.0055
SRR8535251	0.0198	0.0085
SRR3932352	0.0048	0.0021
ERR2619102	0.0072	0.0031

Table 5: The Mash distance between $R^i_{0.6}$ ($R^i_{0.8}$) and R^i_{rep} in 10 different samples. The samples here correspond to those listed in Table 3.

We then empirically examine the differences in the performance of Auto-ParAdvisor with and

Samples	Subsample (0.6)	Subsample (0.8)
SRR3385209	0.0054	0.0023
SRR8274574	0.0076	0.0036
SRR5659428	0.0068	0.0023
SRR4318442	0.0104	0.0043
SRR6312630	0.0106	0.0042
SRR8273981	0.0095	0.0037
SRR5604831	0.0081	0.0034
SRR5878078	0.0057	0.0024
SRR9321872	0.0097	0.0034
SRR1981008	0.0084	0.0037

Table 6: The Mash distance between $R^i_{0.6}$ ($R^i_{0.8}$) and R^i_{rep} in another 10 different samples. Different from samples shown in Table 5, the samples here correspond to those listed in Table 4, which are randomly selected samples from the representative set for StringTie.

without the implementation of data augmentation. We randomly select 150 samples from $\mathcal{R}_{rep}$ to form a validation set, while the remaining samples are used for training. For each sample R in the validation set, we compute the Pearson correlation between the predicted similarity values and the actual predefined similarity values $\hat{\mathbb{S}}_{rep}(R, R')$, where R' represents any sample in the training set. We then compare the average of these correlations across all samples in the validation set, contrasting the results obtained when training with the data augmentation module against those from training without it. Figure 4 shows the average Pearson correlations of all samples in the validation set across various training iterations. It is evident that the inclusion of data augmentation leads to significantly higher Pearson correlation values compared to the scenarios where no data augmentation is used. This trend holds true regardless of the transcript assembler employed. These findings indicate the beneficial impact and importance of the data augmentation module in enhancing the performance of Auto-ParAdvisor.

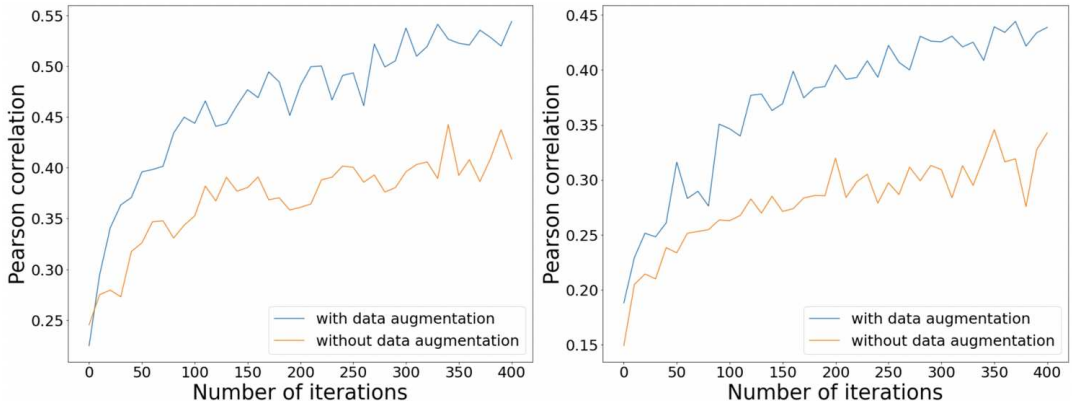

Figure 4: The average Pearson correlations of all samples in the validation set across various training iterations. The transcript assembler used for the left figure is Scallop, while the transcript assembler used for the right figure is StringTie.

4.5 Effectiveness of Advisor sets from Auto-ParAdvisor

4.5.1 ENCODE65

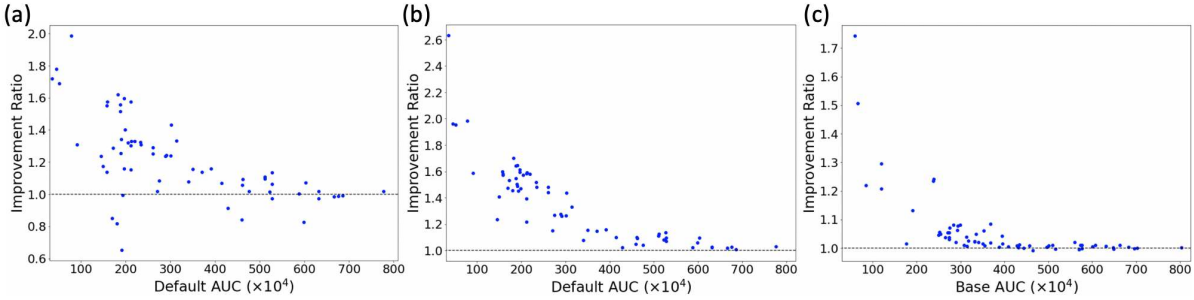

Figure 5: (a,b,c) Comparisons between Auto-ParAdvisor against the default parameter settings or Base in the top-1(a), top-5(b), and top-30(c) cases for ENCODE65. Each point is one RNA-seq sample positioned by the AUC achieved with the default parameters or the best AUC obtained with Base and the ratio of the best AUC achieved by Auto-ParAdvisor relative to the default parameter setting or Base. A value above 1.0 indicates an improvement. The transcript assembler used here is Scallop.

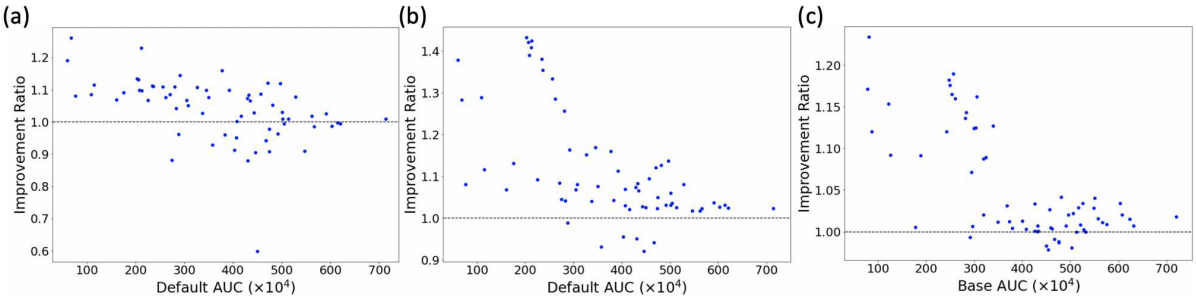

Figure 6: (a,b,c) Comparisons between Auto-ParAdvisor against the default parameter setting or Base in the top-1(a), top-5(b), and top-30(c) cases for ENCODE65. Each point is one RNA-seq sample positioned by the AUC achieved with the default parameters or the best AUC obtained with Base and the ratio of the best AUC achieved by Auto-ParAdvisor relative to the default parameter setting or Base. A value above 1.0 indicates an improvement. The transcript assembler used here is StringTie.

For the ENCODE65 dataset, we compared Auto-ParAdvisor with the parameter advising method for transcript assemblers proposed by DeBlasio et al. [2020], the only existing method of its kind. Their approach generates a fixed advisor set comprising 30 parameter vectors. To ensure a fair comparison, we also generated an advisor set from Auto-ParAdvisor with a size of 30 (i.e. $p = 30$). We use "Base" to represent the method in DeBlasio et al. [2020] in the following. Additionally, we created advisor sets from Auto-ParAdvisor with sizes of 1 and 5 (i.e. $p = 1$ and $p = 5$), comparing them against the default parameter vector. These scenarios are denoted as top-1, top-5, and top-30 cases.

Figures 5 and 6 display the ratios of the best AUC values achieved by Auto-ParAdvisor over the default settings and Base in these three scenarios for the Scallop and StringTie transcript assemblers. In the top-1 case, Auto-ParAdvisor outperforms the default settings in 53 out of 65 samples for Scallop and in 46 out of 65 samples for StringTie. In the top-5 case, Auto-ParAdvisor surpasses the default settings in all samples for Scallop and in 59 out of 65 samples for StringTie. In the top-30 case,

Auto-ParAdvisor outperforms Base in 59 out of 65 samples for Scallop and in 56 out of 65 samples for StringTie. These results collectively highlight the strong performance of Auto-ParAdvisor.

We also compare Auto-ParAdvisor with two simpler methods on the ENCODE65 dataset:

- Direct application of CAWarm-BO on each sample in ENCODE65.
- Implementation of a method called Auto-Mash, which follows the same framework as Auto-ParAdvisor but uses the negative Mash distance, instead of the contrastive learning framework, to compute the similarity measure $\mathbb{S}$ between samples.

The results of these comparisons are displayed in Figures 11 and 12 in the appendix. Generally, Auto-ParAdvisor has superior performance over both CAWarm-BO and Auto-Mash. These findings suggest that directly employing CAWarm-BO on each new sample is not the most efficient approach, and that the contrastive learning framework employed in Auto-ParAdvisor is more effective in learning a better similarity measure $\mathbb{S}$ by leveraging information from the BO process.

4.5.2 SRA-test

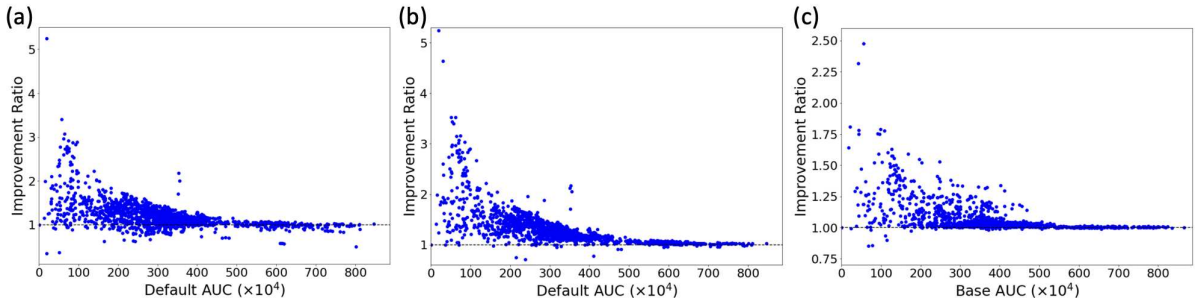

Figure 7: (a,b,c) Comparisons between Auto-ParAdvisor against the default parameter settings or Base in the top-1(a), top-5(b), and top-30(c) cases for SRA-test. Each point is one RNA-seq sample positioned by the AUC achieved with the default parameters or the best AUC obtained with Base and the ratio of the best AUC achieved by Auto-ParAdvisor relative to the default parameter setting or Base. A value above 1.0 indicates an improvement. The transcript assembler used here is Scallop.

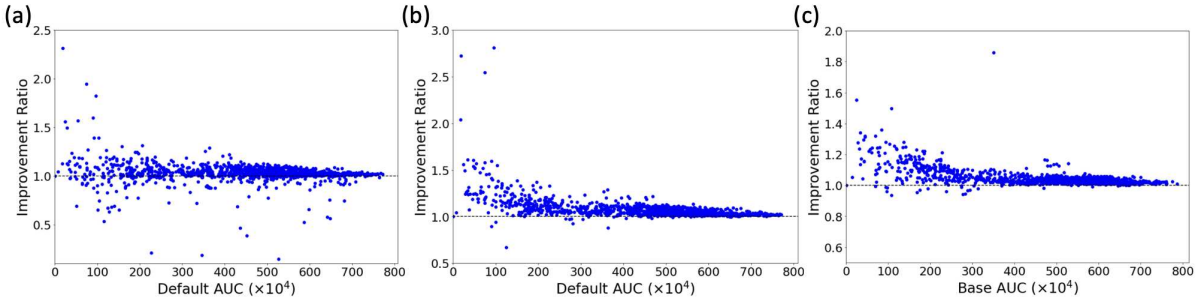

Figure 8: (a,b,c) Comparisons between Auto-ParAdvisor against the default parameter setting or Base in the top-1(a), top-5(b), and top-30(c) cases for SRA-test. Each point is one RNA-seq sample positioned by the AUC achieved with the default parameters or the best AUC obtained with Base and the ratio of the best AUC achieved by Auto-ParAdvisor relative to the default parameter setting or Base. A value above 1.0 indicates an improvement. The transcript assembler used here is StringTie.

For the SRA-test dataset, Figures 7 and 8 demonstrate Auto-ParAdvisor's robust performance on this extensive dataset. In the top-1 case, Auto-ParAdvisor outperforms the default settings in 81.6% of samples for Scallop and 84.9% for StringTie. In the top-5 case, Auto-ParAdvisor surpasses the default settings in 97.0% of samples for Scallop and 97.8% for StringTie. In the top-30 case, Auto-ParAdvisor outperforms Base in 94.9% of samples for Scallop and 97.4% for StringTie.

4.6 Running time of Auto-ParAdvisor

The most time-consuming part of constructing Auto-ParAdvisor for a specific transcript assembler is to search for the optimal parameter vectors for representative samples using BO. In practice, this process requires approximately 15 days using around 300 CPUs to complete BO on all samples in $\mathcal{R}_{rep}$. We leave to future work the investigation of more efficient methods for searching the best parameter vectors for the training set. Creating set representations of samples in the training set and running the data augmentation module usually take 1-2 days with simple CPU parallelism. Once all these steps are completed, the training of our contrastive framework is notably swift, taking only about 10 minutes on one GPU.

As introduced in DeBlasio et al. [2020], for Scallop, running coordinate ascent for any single example took between about 40 hours and over 22 days. Therefore, even with parallelism, the creation of the advisor set as per DeBlasio et al. [2020] can take around 22 days. Thus, in terms of wall clock time, the training of our method is on par with the time required to generate the advisor set of DeBlasio et al. [2020] for Scallop, although it does necessitate greater parallelism and consequently more CPU resources.

After Auto-ParAdvisor has been trained, it eliminates the need for retraining with each new sample, allowing for rapid prediction of the advisor set for new samples. The only additional computational steps for our method are creating the set representation of a new sample using the MinHash sketch technique and computing the nearest neighbors of the new sample. Both of these steps are efficient, typically taking less than 10 minutes to complete.

5 Discussion

We introduce Auto-ParAdvisor, the first parameter advisor that can generate a sample-specific advisor set for each RNA-seq sample, for more accurate transcript assembly. To achieve this, we develop a new Bayesian Optimization method that can efficiently search for the best parameter vectors of representative samples. We also develop a contrastive learning framework for learning a similarity measure between samples and building the nearest-neighbor model to output sample-specific advisor sets. With the transcript assemblers Scallop and StringTie, we show that Auto-ParAdvisor obtains better performance than the previous method on both the ENCODE65 and SRA-test datasets.

The methods developed in this work are not limited to bioinformatics. The way of combining CA and BO provides a new strategy for optimizing black-box functions with unknown search domains,

and the contrastive learning framework provides new insights on solving a special case of contrastive regression problem: both inputs and outputs are not in standard Euclidean space (in our case inputs are RNA-seq samples and outputs are functions), which is a very interesting problem that has not been extensively discussed and for which no previous methods applies.

The heuristic constructions used in this framework, such as the way to define the similarity between two functions, the form of the contrastive loss function, and the architecture of the encoding network can be further improved in the future. Additionally, following DeBlasio et al. [2020], we use AUC to estimate the performance of transcript assemblers by comparing predicted transcripts with known ones. However, this is not a perfect measure as it may unfairly penalize assemblers that generate novel transcripts, which can be of significant interest. Developing better performance metrics is an interesting direction for future work. Finally, although for a specific transcript assembler Auto-ParAdvisor does not need to be retrained for any new RNA-seq sample, it needs to be retrained when the transcript assembler changes. Therefore, developing a method that can transfer knowledge from one transcript assembler to another to speed up this retraining procedure is also an important direction for future work.

Acknowledgements

The authors would like to thank the members of the Kingsford Group for their helpful comments throughout this project, in particular Minh Hoang, Guillaume Marçais, Laura Tung, Dan DeBlasio, and Yutong Qiu. This work was supported in part by the US National Science Foundation [DBI-1937540, III-2232121], the US National Institutes of Health [R01HG012470] and by the generosity of Eric and Wendy Schmidt by recommendation of the Schmidt Futures program. Conflict of Interest: C.K. is a co-founder of Ocean Genomics, Inc.

References

Mingfu Shao and Carl Kingsford. Accurate assembly of transcripts through phase-preserving graph decomposition. *Nature Biotechnology*, 35(12):1167–1169, 2017.

Qimin Zhang, Qian Shi, and Mingfu Shao. Scallop2 enables accurate assembly of multiple-end RNA-seq data. *bioRxiv*, 2021.

Cole Trapnell, Brian A Williams, Geo Pertea, Ali Mortazavi, Gordon Kwan, Marijke J Van Baren, Steven L Salzberg, Barbara J Wold, and Lior Pachter. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology*, 28(5):511–515, 2010.

Mitchell Guttman, Manuel Garber, Joshua Z Levin, Julie Donaghey, James Robinson, Xian Adiconis, Lin Fan, Magdalena J Koziol, Andreas Gnirke, Chad Nusbaum, et al. Ab initio reconstruction

of cell type–specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincRNAs. *Nature Biotechnology*, 28(5):503–510, 2010.

Alexandru I Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. A novel min-cost flow method for estimating transcript expression with RNA-Seq. In *BMC Bioinformatics*, volume 14, page S15. Springer, 2013.

Li Song, Sarven Sabunciyan, and Liliana Florea. CLASS2: accurate and efficient splice variant annotation from RNA-seq reads. *Nucleic Acids Research*, 44(10):e98–e98, 2016.

Juntao Liu, Ting Yu, Tao Jiang, and Guojun Li. TransComb: genome-guided transcriptome assembly via combing junctions in splicing graphs. *Genome Biology*, 17(1):213, 2016.

Mihaela Pertea, Geo M Pertea, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nature Biotechnology*, 33(3):290–295, 2015.

Shunfu Mao, Lior Pachter, David Tse, and Sreeram Kannan. RefShannon: A genome-guided transcriptome assembler using sparse flow decomposition. *PloS one*, 15(6):e0232946, 2020.

Sam Kovaka, Aleksey V Zimin, Geo M Pertea, Roham Razaghi, Steven L Salzberg, and Mihaela Pertea. Transcriptome assembly from long-read RNA-seq alignments with StringTie2. *Genome Biology*, 20(1):278, 2019.

Laura H Tung, Mingfu Shao, and Carl Kingsford. Quantifying the benefit offered by transcript assembly with Scallop-LR on single-molecule long reads. *Genome Biology*, 20(1):287, 2019.

Juntao Liu, Xiangyu Liu, Xianwen Ren, and Guojun Li. scRNAss: a single-cell RNA-seq assembler via imputing dropouts and combing junctions. *Bioinformatics*, 35(21):4264–4271, 2019.

Ka Ming Nip, Readman Chiu, Chen Yang, Justin Chu, Hamid Mohamadi, René L Warren, and Inanc Birol. RNA-Bloom enables reference-free and reference-guided sequence assembly for single-cell transcriptomes. *Genome Research*, 30(8):1191–1200, 2020.

Adam Voshall and Etsuko N Moriyama. Next-generation transcriptome assembly: strategies and performance analysis. *Bioinformatics in the Era of Post Genomics and Big Data*, pages 15–36, 2018.

Dan DeBlasio, Kwanho Kim, and Carl Kingsford. More accurate transcript assembly via parameter advising. *Journal of Computational Biology*, 27(8):1181–1189, 2020.

Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? Generalization guarantees for data-driven algorithm design. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 919–932, 2021.

Luis Cedillo, Hector Richart Ruiz, and Dan DeBlasio. Exploiting large datasets improves accuracy estimation for multiple sequence alignment. *bioRxiv*, 2022.

Dan DeBlasio and John Kececioglu. Learning parameter sets for alignment advising. In *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 230–239, 2014.

Dan DeBlasio and John Kececioglu. Learning parameter-advising sets for multiple sequence alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 14(5):1028–1041, 2015a.

Dan DeBlasio and John Kececioglu. Parameter advising for multiple sequence alignment. In *BMC Bioinformatics*, volume 16, page A3. BioMed Central, 2015b.

Chris S Magnano and Anthony Gitter. Automating parameter selection to avoid implausible biological pathway models. *NPJ Systems Biology and Applications*, 7(1):12, 2021.

Peter I Frazier. A tutorial on Bayesian Optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2015.

Eric Schulz, Maarten Speekenbrink, and Andreas Krause. A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of Mathematical Psychology*, 85:1–16, 2018.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, 25, 2012.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial Intelligence and Statistics*, pages 528–536. PMLR, 2017.

Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

Alonso Marco, Felix Berkenkamp, Philipp Hennig, Angela P Schoellig, Andreas Krause, Stefan Schaal, and Sebastian Trimpe. Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with Bayesian optimization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1557–1563. IEEE, 2017.

Aaron Wilson, Alan Fern, and Prasad Tadepalli. Using trajectory data to improve Bayesian optimization for reinforcement learning. *The Journal of Machine Learning Research*, 15(1):253–282, 2014.

Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. Constrained Bayesian optimization for automatic chemical design using variational autoencoders. *Chemical Science*, 11(2):577–586, 2020.

Diana M Negoescu, Peter I Frazier, and Warren B Powell. The knowledge-gradient algorithm for sequencing experiments in drug discovery. *INFORMS Journal on Computing*, 23(3):346–363, 2011.

Phuc H Le-Khac, Graham Healy, and Alan F Smeaton. Contrastive representation learning: A framework and review. *IEEE Access*, 8:193907–193934, 2020.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in Neural Information Processing Systems*, 33:18661–18673, 2020.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, pages 1597–1607. PMLR, 2020.

Ching-Yao Chuang, Joshua Robinson, Yen-Chen Lin, Antonio Torralba, and Stefanie Jegelka. Debiased contrastive learning. *Advances in Neural Information Processing Systems*, 33:8765–8775, 2020.

Yaoming Wang, Yangzhou Jiang, Jin Li, Bingbing Ni, Wenrui Dai, Chenglin Li, Hongkai Xiong, and Teng Li. Contrastive regression for domain adaptation on gaze estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19376–19385, 2022.

Pengfei Zhang, Zhengyuan Jiang, Yixuan Wang, and Yu Li. CLMB: deep contrastive learning for robust metagenomic binning. In *International Conference on Research in Computational Molecular Biology*, pages 326–348. Springer, 2022.

Laura H Tung and Carl Kingsford. Practical selection of representative sets of RNA-seq samples using a hierarchical approach. *Bioinformatics*, 37(Supplement_1):i334–i341, 2021.

Jacob M Schreiber, Jeffrey A Bilmes, and William Stafford Noble. apricot: Submodular selection for data summarization in Python. *J. Mach. Learn. Res.*, 21(161):1–6, 2020.

Bobak Shahriari, Alexandre Bouchard-Côté, and Nando Freitas. Unbounded Bayesian optimization via regularization. In *Artificial Intelligence and Statistics*, pages 1168–1176. PMLR, 2016.

Matthias Poloczek, Jialei Wang, and Peter I Frazier. Warm starting Bayesian optimization. In *2016 Winter Simulation Conference (WSC)*, pages 770–781. IEEE, 2016.

Willard I Zangwill. *Nonlinear programming: a unified approach*, volume 52. Prentice-Hall Englewood Cliffs, NJ, 1969.

Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.

Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology*, 17(1):132, 2016.

Annick Lesne, Julien Riposo, Paul Roger, Axel Cournac, and Julien Mozziconacci. 3D genome reconstruction from chromosomal contacts. *Nature Methods*, 11(11):1141–1143, 2014.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in Neural Information Processing Systems*, 30, 2017.

Xingchen Wan, Vu Nguyen, Huong Ha, Binxin Ru, Cong Lu, and Michael A Osborne. Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces. *arXiv preprint arXiv:2102.07188*, 2021.

Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

The ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414):57, 2012.

Guy Cochrane, Ilene Karsch-Mizrachi, Toshihisa Takagi, and International Nucleotide Sequence Database Collaboration. The international nucleotide sequence database collaboration. *Nucleic Acids Research*, 44(D1):D48–D50, 2016.

David Salinas, Huibin Shen, and Valerio Perrone. A quantile-based approach for hyperparameter transfer learning. In *International Conference on Machine Learning*, pages 8438–8448. PMLR, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. pmlr, 2015.

Diederik P Kingma and Jimmy Ba. ADAM: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A Other implementation details

A.1 Details of CAWarm-BO

In Scallop, we adjust 18 non-negative parameters, and in StringTie, we tune 8 parameters, which is the same as DeBlasio et al. [2020]. The names of these parameters, their default settings and their types are listed in Table 7 and Table 8. The initial range for each non-binary parameter is defined as $[0, 10 \times \text{default}]$ (e.g. for max_dp_table_size in Table 7, the initial range includes all integers within $[0, 100000]$). CAWarm-BO initially runs the same coordinate ascent algorithm as proposed by DeBlasio et al. [2020], with a maximum warmup iteration set at 60. The step size for each coordinate in the ascent is determined by the initial step size specified by DeBlasio et al. [2020]. The warmup phase of CAWarm-BO will cease if it exhaustively interrogates the entire parameter set without any improvement or upon reaching the maximum warmup iteration.

Assuming the best parameter vector identified during warmup is $\hat{\theta}$, where $\hat{\theta}[i]$ represents the value of its i-th dimension, CAWarm-BO adjusts the search range of the i-th dimension to $[0, \max(2\hat{\theta}[i], U_i)]$, with $[0, U_i]$ being the initial range of the i-th dimension. Following this, CAWarm-BO executes CASMOPOLITAN over this new domain. The total number of iterations for CAWarm-BO per sample is set to 200, inclusive of the warmup phase.

Parameter	Default value	Type
uniquely_mapped_only	False	binary
use_second_alignment	False	binary
max_dp_table_size	10000	integer
max_edit_distance	10	integer
max_num_exons	1000	integer
min_bundle_gap	50	integer
min_exon_length	20	integer
min_flank_length	3	integer
min_mapping_quality	1	integer
min_num_hits_in_bundle	20	integer
min_router_count	1	integer
min_splice_boundary_hits	1	integer
min_subregion_gap	3	integer
min_subregion_length	15	integer
min_transcript_length_base	150	integer
min_transcript_length_increase	50	integer
max_intron_contamination_coverage	2	float
min_subregion_overlap	1.5	float

Table 7: Parameter information of Scallop.

A.2 Details of contrastive learning

As described in Section 3.4.1, we use MinHash sketch to create the set input $X = \{x_i\}_{i=1}^{s}$ for each RNA-seq sample R. Each value in each x_i is z-score normalized. We set $k = 21$ and $s = 1000$ which are default settings in Ondov et al. [2016] and exclude k-mers with counts less than 2.

Parameter	Default value	Type
t	False	binary
u	False	binary
m	200	integer
a	10	integer
g	50	integer
j	1	float
f	0.01	float
M	31	float

Table 8: Parameter information of StringTie.

The architecture of h_φ is a fully connected neural network with layer sizes $(2, 128, 128, 128, 128)$. Meanwhile, the architecture of the network g_ψ is another fully connected neural network with layer sizes $(128, 256, 256, 256, 64)$. For each layer except the last one in both networks, we use the ReLU activation function, coupled with batch normalization [Ioffe and Szegedy, 2015] preceding each activation. We use ADAM [Kingma and Ba, 2014] with a learning rate 1×10^{-3} to optimize the neural network. The batch size in each iteration is 128 and the total number of epochs is 400. The training is conducted using one NVIDIA GeForce RTX 3080 Ti GPU.

B Additional experimental results

Samples	CAWarm-BO	CASMOPOLITAN	CA
SRR307903	611.70 (165)	610.98 (146)	613.12 (658)
SRR307911	545.20 (156)	543.35 (198)	549.39 (736)
SRR315323	411.25 (176)	409.61 (150)	409.29 (458)
SRR315334	576.45 (173)	576.37 (156)	579.69 (460)
SRR387661	298.49 (177)	299.53 (188)	296.57 (517)
SRR534291	534.00 (171)	513.68 (200)	534.17 (458)
SRR534307	732.26 (196)	729.79 (182)	733.01 (639)
SRR534319	318.37 (185)	317.87 (96)	315.98 (290)
SRR545695	395.08 (180)	388.43 (80)	393.33 (447)
SRR545723	551.56 (174)	552.10 (144)	552.81 (372)

Table 9: Comparisons of Coordinate Ascent (CA), CASMOPOLITAN, and CAWarm-BO on ENCODE10 dataset. The transcript assembler used here is Scallop. Both CASMOPOLITAN, and CAWarm-BO are run within 200 iterations, while we run Coordinate Ascent until the stopping requirement set by DeBlasio et al. [2020] is satisfied. The table shows the best AUC value ($\times 10^4$) found for each method and the number of iterations (in bracket) used for finding the best value.

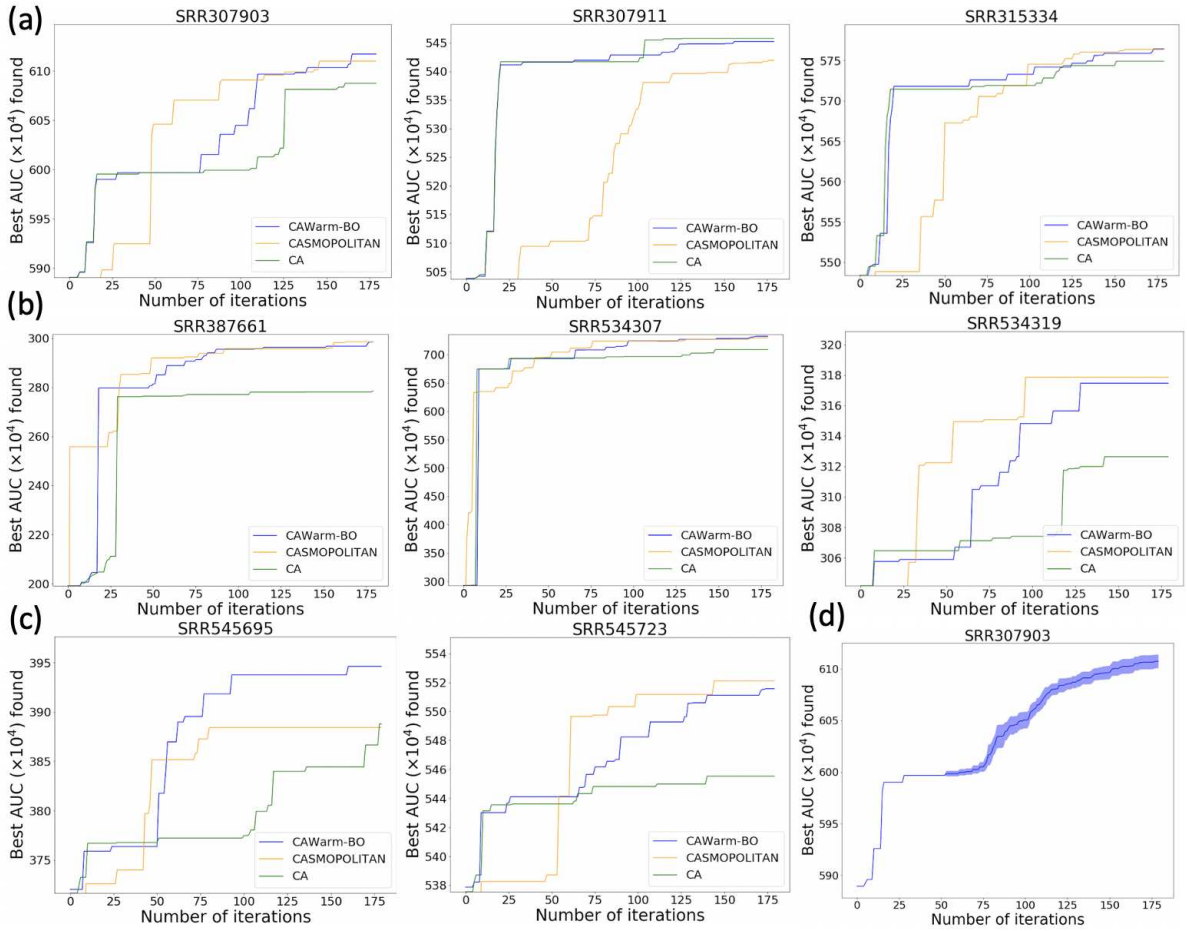

Figure 9: (a,b,c) Comparisons of Coordinate Ascent (CA), CASMOPOLITAN, and CAWarm-BO on eight RNA-seq samples from ENCODE10. The transcript assembler used here is Scallop. For each sample, we do one run for each method and plot the maximum value found by iterations. (d) The variance of CAWarm-BO with different runs. Here, we do 10 independent runs on the sample SRR307903, and plot the mean and 1/2 standard deviation of the maximum value found by iterations. Note that the warm-up part is deterministic.

Samples	CAWarm-BO	CASMOPOLITAN	CA
SRR307903	542.26 (186)	541.50 (115)	542.59 (387)
SRR307911	478.17 (191)	478.52 (173)	474.37 (328)
SRR315323	361.80 (174)	358.15 (149)	361.74 (289)
SRR315334	488.93 (195)	489.20 (163)	489.54 (302)
SRR387661	441.58 (182)	441.57 (165)	442.02 (274)
SRR534291	465.90 (165)	463.69 (140)	466.04 (225)
SRR534307	634.86 (173)	635.65 (166)	635.94 (316)
SRR534319	259.91 (197)	260.39 (174)	262.69 (304)
SRR545695	338.88 (181)	333.90 (168)	338.85 (303)
SRR545723	501.54 (178)	500.47 (162)	502.03 (292)

Table 10: Comparisons of Coordinate Ascent (CA), CASMOPOLITAN, and CAWarm-BO on ENCODE10 dataset. The transcript assembler used here is StringTie. Both CASMOPOLITAN, and CAWarm-BO are run within 200 iterations, while we run Coordinate Ascent until the stopping requirement set by DeBlasio et al. [2020] is satisfied. The table shows the best AUC value ($\times 10^4$) found for each method and the number of iterations (in bracket) used for finding the best value.

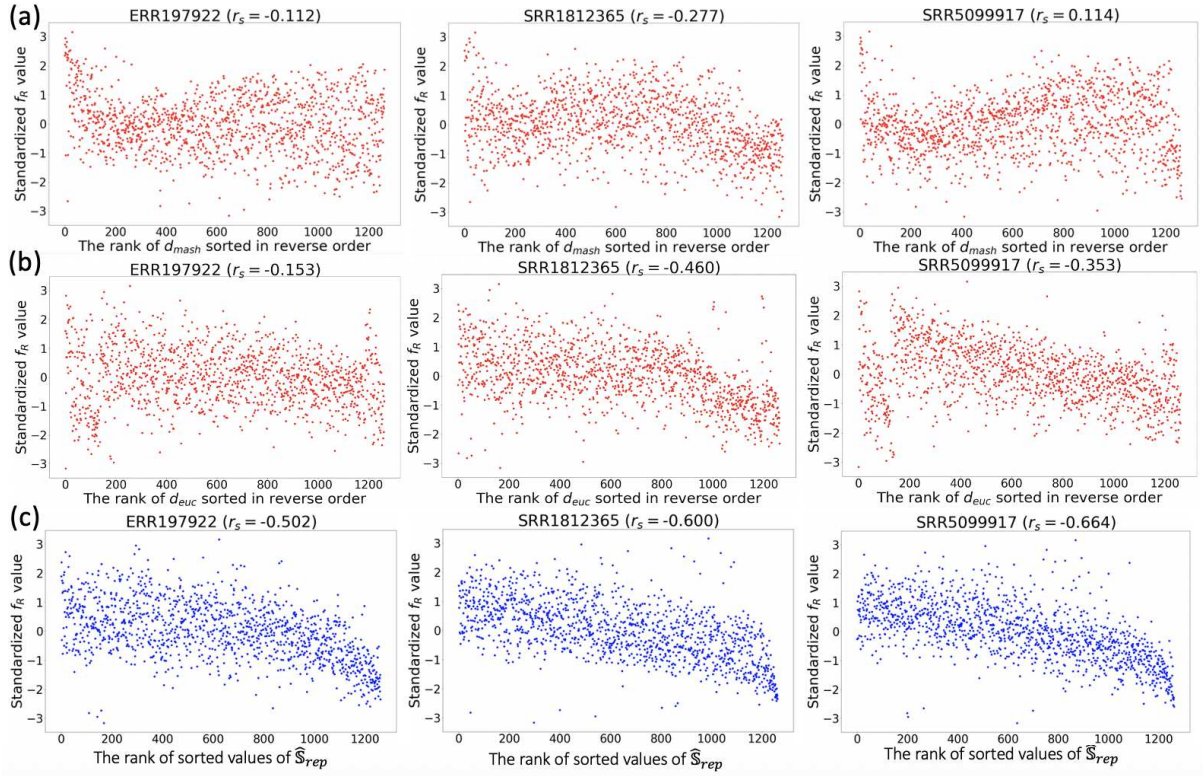

Figure 10: The correlation between (a) the rank of Mash similarity (negative Mash distance values $-d_{mash}$), (b) the rank of Euclidean similarity (negative Euclidean distance values $-d_{euc}$) or (c) the rank of sorted values of $\hat{\mathbb{S}}_{rep}$ and function values $\{f_R(\hat{\theta}^i)\}_{i=1}^m$ for three RNA-seq samples with accession numbers ERR197922, SRR1812365, SRR5099917. Each dot represents one representative sample (one $\hat{\theta}^i$). The transcript assembler $\mathcal{A}$ used here is Scallop. Each y-axis is copula standardized [Salinas et al., 2020] for better visualization. r_s: Spearman's rank correlation coefficient.

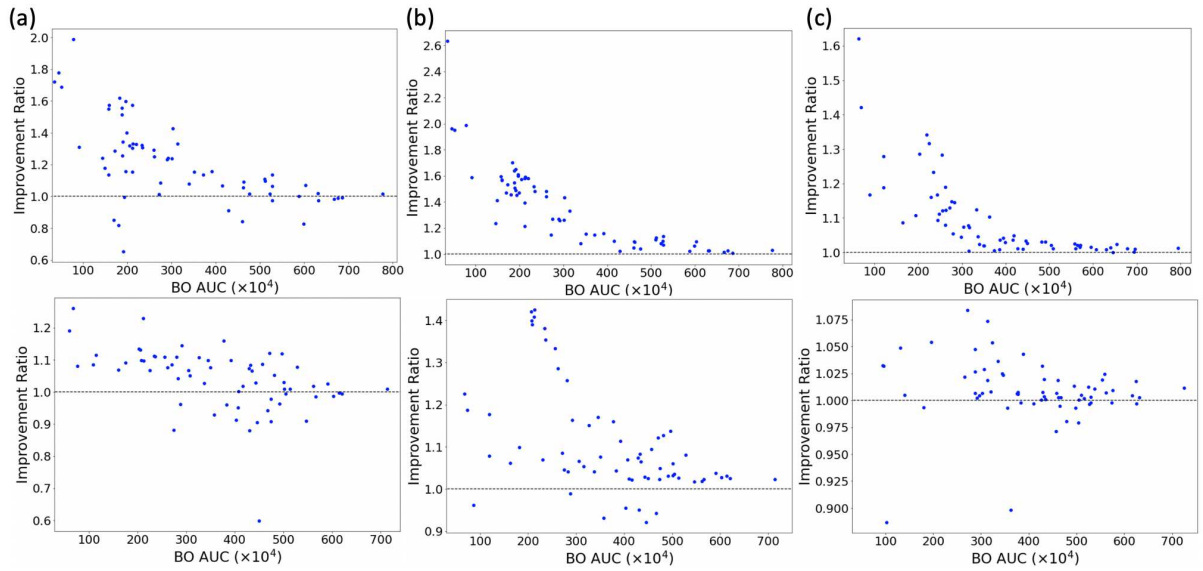

Figure 11: (a,b,c) Comparisons between Auto-ParAdvisor against direct application of CAWarm-BO in the top-1(a), top-5(b), and top-30(c) cases for ENCODE65. Each point is one RNA-seq sample positioned by the best AUC achieved by CAWarm-BO and the ratio of the best AUC achieved by Auto-ParAdvisor relative to CAWarm-BO. A value above 1.0 indicates an improvement. The transcript assembler used for the first row is Scallop and StringTie for the second row.

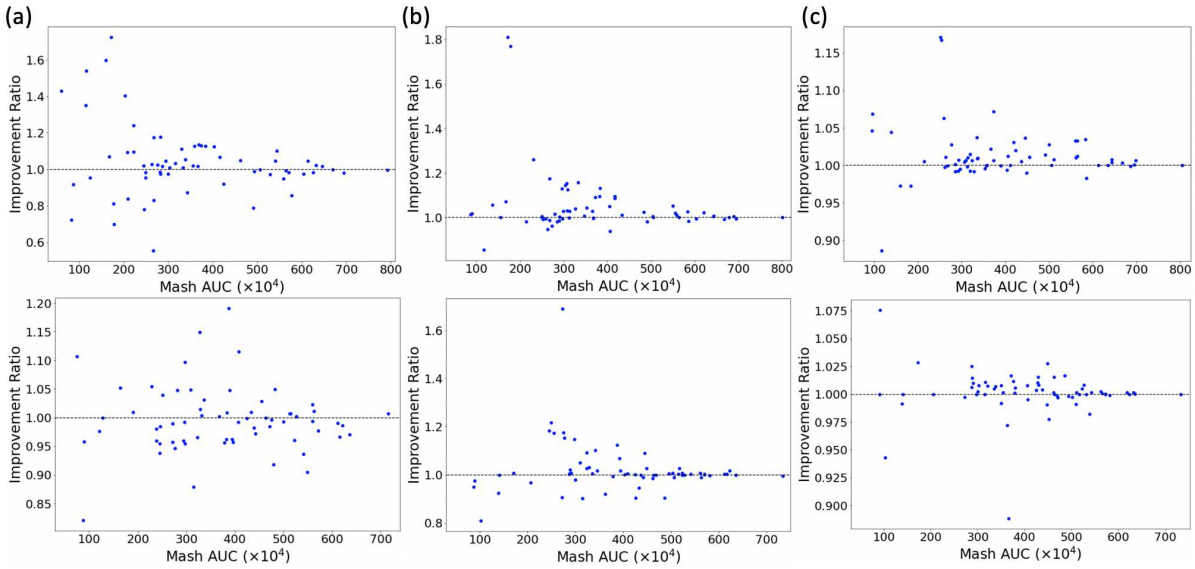

Figure 12: (a,b,c) Comparisons between Auto-ParAdvisor against Auto-Mash in the top-1(a), top-5(b), and top-30(c) cases for ENCODE65. Each point is one RNA-seq sample positioned by the best AUC achieved by Auto-Mash and the ratio of the best AUC achieved by Auto-ParAdvisor relative to Auto-Mash. A value above 1.0 indicates an improvement. The transcript assembler used for the first row is Scallop and StringTie for the second row.