Thesis Proposal

Practical Methods for Automated Algorithm Design in Machine Learning and Computational Biology

Minh Hoang

May 2022

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Prof. Carl Kingsford, Chair
Prof. David P. Woodruff
Prof. Maria-Florina Balcan
Prof. Risto Miikkulainen

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

Configuration tuning is often a key element in achieving outstanding performance with parameterized algorithmic solutions. However, it often requires extensive manual effort to satisfactorily configure an algorithm for a specific task instance, thus preventing these algorithms to be deployed at scale. More importantly, without a principled method to configure these deployment settings, it will be difficult to reproduce the obtained results under other experimental conditions. To address these problems, this thesis focuses on developing novel automated algorithm design (AAD) frameworks capable of configuring algorithms for specific use cases in a data-driven manner. Particularly, we cast these AAD problems as optimization tasks that aim to maximize some performance metric with respect to the configurations of the solution model. We ground our investigation in three specific classes of AAD problems, including kernel selection for Bayesian inference, architecture search for deep neural network and minimizer construction for sequence sketching. In all of these problems, the variables to be optimized often have underlying discrete structures such as trees, graphs or permutations. Our contribution is a suite of reformulation techniques that result in efficient and accurate tuning methods for these configuration domains. Finally, we demonstrate the performance of our methods on practical scenarios and show that they have significantly outperformed state-of-the-art benchmarks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Automated Algorithm Design

A vast majority of algorithms are typically developed with parameters that can be configured by users depending on their use cases. Although it is convenient to use a default configuration for every application, such a tactic is often sub-optimal when performance is sensitive to the choice of configurations[3, 77]. In Bayesian statistics, choosing an appropriate kernel function to model data correlation can strongly influence the outcome of probabilistic inference [15]. In deep learning, the design of a neural network architecture (e.g., the number of layers, types of activation functions and number of neurons in each layer) must be carefully selected to achieve optimal performance on specific scenarios [26, 68, 87]. For example, convolutional neural network architectures that excel on computer vision tasks [27, 50] might underperform in language processing problems, which have been found to favor attention-based modelling [78].

Due to this inconsistent behavior, a more practical approach is therefore to calibrate the design of an algorithm on a *per-task* basis, which seeks to find the most suitable model configuration for each new problem instance. Despite the potential improvements that optimally configuring an algorithm can have on its performance, this calibration step is traditionally driven by domain expert experience and manual heuristics, and hence inefficient and difficult to scale or reproduce. This absence of a principled approach for configuring algorithmic solutions has motivated the study of automated algorithm design

(AAD) through systematic optimization frameworks, which was first considered in the seminal work of Rice [71] and subsequently in various algorithmic domains such as deep learning [26, 37, 68], non-parametric Bayesian methods [15, 57, 58] and discrete algorithms [85, 86]. Formally, the general AAD problem is described as follows:

Definition 1 (Automated Algorithm Design) *Let $\mathcal{T}$ be an arbitrary space of computational tasks and $\mathcal{M}$ be a likely infinite set of algorithms capable of solving any task $\tau \in \mathcal{T}$. Given a performance evaluation function $F : \mathcal{T} \times \mathcal{M} \rightarrow \mathbb{R}$, we say that an algorithm $m \in \mathcal{M}$ outperforms $m' \in \mathcal{M}$ on some task $\tau \in \mathcal{T}$ if and only if $F(\tau, m) > F(\tau, m')$. Then, for any task $\tau \in \mathcal{T}$, the AAD problem can be written as the following optimization task, which seeks to find the optimal algorithm $m_* \in \mathcal{M}$ such that $F(\tau, m')$ is maximized:*

$$m_* \in \operatorname*{argmax}_{m \in \mathcal{M}} F(\tau, m) . \tag{1.1}$$

1.2 Related Work

Due to the general nature of the performance measuring function F, the AAD task in Eq. (1.1) is commonly viewed as a black-box optimization (BBO) problem, where F is taken for an oracle that can be queried at will given some input configuration. Most existing black-box optimization methods tend to approach this task via a sequential optimization strategy which alternates between evaluating observations and making informed decision about subsequent probings of the oracle. Typically, this decision is either guided by practical heuristics or a surrogate model that estimates the relationship between configurations and evaluated performances. We summarize a few typical BBO methods below:

Heuristic search

One of the most classical approach to optimize this black-box function is grid search, which systematically evaluates all input configurations projected on a lattice to find the best performing candidate. For continuous configurations, this lattice is obtained by discretizing the search space, whereas for categorical configurations, this routine simply means exhaustively trying all combinations of values in each dimension. Alternatively, another widely-used approach for black-box optimization is random search, where inputs are chosen completely at random to evaluate. Both grid search and random search are straightforward to implement and have been used in several AAD tasks [7, 8, 53]. These methods, however, only focus on exploring the search space and have no built-in mechanism to exploit the collected observations, thus tend to scale poorly with the complexity of the input domain.

In contrast, other heuristic optimization methods such as coordinate ascent [23, 80] and simulated annealing [9] focus on refining the best candidate found so far in a greedy manner. Coordinate ascent achieves this by successively optimizing along a specific dimension while fixing every other dimension of the parameter space, repeating for all dimensions until convergence. Instead of making queries along a single coordinate, the simulated annealing algorithm alternatively evaluates a small neighborhood around the current best estimate and tries to move in the direction that yield the best improvement. Both methods have also been used in many AAD tasks [5, 8, 13], but are typically vulnerable to being trapped in local optima due to a lack of exploration mechanism.

Evolutionary strategies

Evolutionary strategies (ES) are optimization techniques inspired by nature, in which a population of configurations is set to evolve over time and improve its averaged performance while doing so. For every generation of this population, most evolutionary algorithms will conduct the following three steps: (1) estimate the fitness of each individual in the population; (2) generate new individuals using certain reproduction and/or mutation operators; and (3) replacing unfit individuals with new individuals.

Evolutionary algorithms [4, 22] have been widely applied in automated algorithm design, such as model selection for deep neural networks [52, 56, 63, 66] and support vector machines (SVM) [54]; or finding clinical interventions [64]. These techniques fundamentally differ from the above heuristic algorithms by balancing between exploration and exploitation. For example, in genetic optimization, high-fitness individuals can partially pass down their representations to the next generation via a crossover operator (i.e., exploitation), in hope that good performances can be preserved and improved. At the same time, new representations are continually generated via the random mutation operator (i.e., exploration), thus ensuring the optimization is not trapped in local optima.

Sequential model-based optimization

Unlike the above methods which are considered model-free, sequential model-based optimization (SMBO) is an optimization paradigm which iteratively uses collected information to estimate a surrogate model for the black-box function F. This model is then used to guide the acquisition of subsequent observations. Different SMBO methods have been used to address various AAD tasks such as configuring parameterized tree search algorithms and local SAT solvers [36]. We now give a description of the Bayesian optimization (BO) algorithm [75], which is a widely-used variant of SMBO.

The general BO algorithm usually prescribes a Gaussian Process (GP) prior [70] over the black-box objective function, i.e. $F \sim \mathcal{GP}(\mu, k)$ where $\mu : \mathcal{M} \rightarrow \mathbb{R}$ and $k : \mathcal{M} \times \mathcal{M} \rightarrow \mathbb{R}$ are respectively the prior GP mean and covariance functions. This prior implies that for any finite subset of candidate configurations $\{m_1, m_2 \ldots m_T\}$ the corresponding performance vector $[F(m_1)F(m_2)\ldots F(m_T)]$ is normally distributed *a priori* with mean $[\mu(m_1), \mu(m_2), \ldots, \mu(m_T)]$ and covariance $[k(m_i, m_j)]_{i,j\in[T]}$. At any iteration t, the BO algorithm then uses this prior distribution and the set $\mathcal{D}_t$ of collected observations so far to derive a posterior predictive distribution $p(F(m_*) \mid m_*, \mathcal{D}_t)$ for any subsequent candidate m_*.

The posterior predictive mean can be used directly to estimate the expected performance of subsequent candidates, thus allowing us to exploit high-performing configurations without actually evaluating F. However, since the posterior naturally has high uncertainty in unobserved regions, doing so would discourage thorough exploration of the search space and risk missing out on good solutions. To address this issue, the BO algorithm instead constructs an *acquisition function* that incorporates both the posterior mean and covariance to balance this exploitation and exploration trade-off. For example, the upper confidence bound (UCB) acquisition function proposed by Srinivas et al. [76] is given by:

$$\alpha_{\text{UCB}}(m; M_t, \sigma_t) \quad \triangleq \quad M_t(m) + \beta\sqrt{\sigma_t(m)}\,, \tag{1.2}$$

where $M_t(m)$ and $\sigma_t(m)$ respectively denote the posterior mean and variance at iteration t given some candidate m. Here, the parameter β reflects the trade-off between exploiting candidates with high expected performance and exploring candidates with high uncertainty. Finally, the BO algorithm can be described via the following update rules:

$$\begin{aligned} m_{t+1} &= \arg\max_{m \in \mathcal{M}} \alpha_{\mathrm{UCB}}\left(m; M_t, \sigma_t\right) \ , \\ \mathcal{D}_{t+1} &= \mathcal{D}_t \cup \left\{(m_{t+1}, F(m_{t+1})\right\} \ , \\ M_{t+1}, \sigma_{t+1} &\leftarrow p\left(F(m_*) \mid m_*, \mathcal{D}_{t+1}\right) \ . \end{aligned} \tag{1.3}$$

Nonetheless, we remark that the vanilla BO algorithm is best suited for low-dimensional and continuous input domains. In practical AAD tasks where the space of configuration is structured and discrete, there is generally no unifying approach to parameterize the mean and covariance function of the GP surrogate. Furthermore, optimizing the acquisition functions will also become non-trivial and require special modelling considerations.

Domain-specific AAD approaches

In many specific AAD problems, $\mathcal{M}$ can be specified as a sub-class of algorithms that is described by discrete data structures such as sub-trees [15], graphs [6, 26, 68] or permutations [59]. This choice of representation implicitly prescribes a correlation structure among candidate models via the topology of $\mathcal{M}$ and thus allows the general AAD objective to subsequently be cast as optimization/search tasks on these structured domains.

This thesis will focus on three such prototypical classes of AAD tasks that can be unified through the lens of structured optimization, namely kernel selection (KS), minimizer sketch design (MSD) and neural architecture search (NAS). In particular, the KS problem in Bayesian statistics [15, 58] can be formulated as a tree search routine to find the composite function that best models the covariance structure of a stochastic process. The NAS problem is approached via setting $\mathcal{M}$ to be a set of computation graphs that share the same vertex and edge space, which respectively denotes all intermediate feature representations and all possible transformations [68, 87]. Lastly, the MSD problem, which seeks to find an optimal sketching scheme for biological sequences, is expressed as finding the optimal permutation of all fixed length substrings induced by the sequence vocabulary [59, 85, 86].

Even though this domain restriction strategy has enabled more meaningful formulations of the AAD problem, their resulting discrete/combinatorial objectives are still challenging to solve due to the prohibitive sizes of their search domains. Concretely, without any further restriction, the tree search space in the KS problem naturally has unbounded depth, whereas the graph and permutation domains of the MSD/NAS problems are virtually infinite in most practical settings.

In practice, this large amount of admissible candidate configurations would render the majority of standard approaches such as integer programming [1, 2] and heuristic search inefficient, and thus necessitates better informed search strategies to navigate these massive search spaces. In addition, another major challenge arises due to the fact that none of the respective performance measuring functions have a closed-form expression, nor are they computationally cheap to evaluate. For example, measuring the fitness of a neural network architecture would typically entail training all layer parameters to convergence before measuring its predictive loss/accuracy with respect to some validation dataset. As such, many strategies that rely on repeatedly probing this evaluation function are inefficient and not applicable in practice.

To address these challenges, a significant portion of the existing AAD literature is centered around the idea of developing an efficient knowledge transfer mechanism that can generalize algorithmic behaviors across different configuration instances. In particular, given some existing observed performance evaluations, methods under this paradigm often aim to (a) infer unseen regions of high-performing configurations; or (b) reduce the cost of evaluating new candidates. Examples of the former include sequential model-based optimization approaches which iteratively refine a surrogate performance function and leverage this to guide the acquisition of new observations [43, 57].

On the other hand, the latter direction focuses on making structural assumptions about the search space to facilitate reusing knowledge from past evaluations. A typical example of this approach is the weight sharing scheme in neural architecture search [26, 68], where every candidate architecture is constructed from the same pool of building-block layers. Each layer in this search space is continually optimized throughout all search iterations (i.e., whenever selected by the search algorithm), thus will alleviate the need to re-train each architecture from scratch. For ease of exposition, we will defer the review of these domain-specific methods to their corresponding technical chapter.

1.3 Research goal and contributions

Most approaches in the various AAD domains that we investigate (i.e., KS, MSD and NAS) either still suffer from the scalability issues above; or must rely on additional domain restrictions to sufficiently prune their massive search spaces. For instance, existing KS methods [15, 57, 58] typically require that all candidate kernel functions must have upper-bounded complexities (e.g., functions with short expressions), whereas the permutation domain in the MSD problem is predominantly approximated by the space of sparse hitting sets [18, 85]. In practice, while such assumptions serve to ensure the feasibility of their respective optimization objectives, they are typically heuristics that do not factor in the optimalities of the pruned candidates. In addition, we note that most AAD frameworks only focus on exploiting the observed performances of configurations in the same task domain (i.e., fixing τ in Eq. (1.1)). In practice, this is not always the sole source of information to guide exploration of the candidate space. For instance, in applications that require tuning for many related tasks, it would be more intuitive to leverage the combined knowledge from all task domains, rather than to independently approach these problems with their respective observations. Motivated by these shortcomings, this thesis therefore seeks to address the following research question: **Can we design information-efficient frameworks that can better exploit both intra-task and inter-task information, thus improving the scalability and performance for automated algorithm design?**

Specifically, this thesis proposes two new directions for information-efficient AAD frameworks, which are grounded in the settings of three practical AAD problems, namely kernel selection (KS), minimizer sketch design (MSD), and neural architecture search (NAS). The first direction seeks to improve the exploitation of intra-task information through learning complementary components that can reason about the desirability of candidate configurations without fully committing expensive performance evaluations. In the context of the KS problem, this manifests as an early-stopping policy for a recurrent kernel generator. This policy outputs the optimal pruning point in an infinitely long generative trajectory of kernel expressions (Chapter 2). For the MSD problem, this component takes form as a template model that learns to generate desirable substring ranking patterns which can subsequently be refined into high-performing candidate permutations (Chapter 3). In both settings, we show that these are more effective formulations of the AAD objective that yield better performing configurations than other state-of-the-art methods.

Orthogonal to the above approaches, the second direction further investigates a *multi-task* knowledge transfer paradigm that leverages information sharing across *different tasks* to improve the performance and efficiency of concurrent AAD instances. Specifically, we study an extension of the neural architecture search problem in a federated learning setting [61], where an ensemble of predictive tasks collaborate to find their respective optimal architectures (Chapter 4). Finally, we propose to investigate a meta-learning approach [21] that can effectively leverage existing model configuration experiences to address unseen and heterogeneous AAD instances, thus improving the scalability of AAD in multi-task learning settings (Chapter 5). The remainder of this thesis will be organized as follows:

- Chapter 2 investigates the kernel selection (KS) problem, which restricts the candidate domain to an infinite-depth tree induced by the kernel composition language [15]. Our contribution is a novel Bayesian optimization framework that optimizes a recurrent kernel generator that outputs infinitely long sequences of kernel expressions. We then jointly learn a termination policy model which decides the optimal stopping point along each infinite generative trajectory. We show that this strategy results in better performing kernel functions on a wide range of predictive tasks.
- Chapter 3 studies the minimizer sketch design (MSD) problem, which aims to find a permutation of substrings that induces the optimal sketch of a sequence via the minimizer algorithm [59, 73]. Our contribution is a novel optimization framework, called DEEPMINIMIZER, which formulates as a pair of substring ranking networks which guarantee different properties of a good solution. Through negotiating for a consensus outcome, these networks result in a continuous relaxation of the original discrete permutation learning objective. We show that our method significantly improves the state-of-the-art performance and scalability of MSD. We further extend our algorithm to unify and provide sketch designs for other sequence sketching methods, such as the syncmer approach [16, 74].
- Chapter 4 investigates the neural architecture search (NAS) problem in a federated learning setting [61], which seeks to concurrently optimize the solution architectures for multiple related tasks without exposing their private data. Our contribution is a novel personalized learning objective which distills the aggregated problem solving experiences into hierarchical architecture components which can be fine-tuned to solve each specific task. We show that this knowledge sharing

scheme is capable of maintaining a high degree of personalization in the solution ensemble and thus significantly improves the efficiency of multi-task neural architecture search.

- Chapter 5 discusses a future work which will extend the *multi-task* AAD paradigm to deal with unseen and heterogeneous tasks. Specifically, drawing inspiration from the meta-learning paradigm for multi-task scenarios, we aim to learn a common meta-configuration that can be adapted quickly to solve any specific task. Unlike the standard meta-learning setting, we consider the scenario where the AAD task distribution in question is assumed to be multi-modal, hence there might not be a single configuration that suits every task. To address this issue, we propose extending the meta-learning framework with population-based optimization to train an evolving ensemble of meta-configurations, which reflects a more fine-grained hierarchical clustering of the diverse task space.

Chapter 2

Kernel Selection

2.1 Problem setting

For ease of exposition, we will ground our study in the context of the multivariate regression problem $y = g(\mathbf{x}) + \epsilon$ given real-valued observation tuples $\mathcal{D} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^{d+1}\}_{i \in [N]}$, where d is the input dimension, ϵ denotes the observation noise and g is the latent function to be inferred. Typically, the prior distribution of g will be modelled such that a tractable posterior predictive distribution $p(g(\mathbf{x}_*) \mid \mathbf{x}_*, \mathcal{D})$ can be analytically derived. For example, in Gaussian process (GP) regression [70], if g is distributed by a GP prior [70] with zero mean and covariance function $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$, such that for any finite subset of inputs $\{\mathbf{x}_{i_1}, \mathbf{x}_{i_2} \ldots \mathbf{x}_{i_m}\}$ where $i_1, i_2 \ldots i_m \in [N]$, we have:

$$\begin{bmatrix} g(\mathbf{x}_{i_1}) \\ g(\mathbf{x}_{i_2}) \\ \cdots \\ g(\mathbf{x}_{i_m}) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix}, \begin{bmatrix} k(\mathbf{x}_{i_1}, \mathbf{x}_{i_1}) & k(\mathbf{x}_{i_1}, \mathbf{x}_{i_2}) & \ldots & k(\mathbf{x}_{i_1}, \mathbf{x}_{i_m}) \\ k(\mathbf{x}_{i_2}, \mathbf{x}_{i_1}) & k(\mathbf{x}_{i_2}, \mathbf{x}_{i_2}) & \ldots & k(\mathbf{x}_{i_2}, \mathbf{x}_{i_m}) \\ \cdots & \cdots & \cdots & \cdots \\ k(\mathbf{x}_{i_m}, \mathbf{x}_{i_1}) & k(\mathbf{x}_{i_m}, \mathbf{x}_{i_2}) & \ldots & k(\mathbf{x}_{i_m}, \mathbf{x}_{i_m}) \end{bmatrix}\right) .$$

If the observation noise ϵ is Gaussian, then the posterior distribution $p(g(\mathbf{x}_*) \mid \mathbf{x}_*, \mathcal{D})$ is also Gaussian and can be tractably derived. This modelling choice of the covariance matrix, broadly referred to as the *kernel trick* [34], is widely used in the GP literature as well as many other probabilistic methods to provide non-linear transformations of data onto some high-dimensional feature space. However, since covariance matrices are required to be positive semi-definite, not all functions will suffice as a *kernel function* in the above formulation. The formal condition for a kernel function k to be valid is given as follows:

Definition 2 (Valid Kernel) *A kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is valid if and only if for any subset of inputs $\{\mathbf{x}_{i1}, \mathbf{x}_{i2} \ldots \mathbf{x}_{im}\}$ and for all $\gamma \in \mathbb{R}^m$ we have:*

$$\sum_{u=1}^{m}\sum_{v=1}^{m} \gamma_u \gamma_v \cdot k(\mathbf{x}_{iu}, \mathbf{x}_{iv}) \quad \geq \quad 0 \,, \tag{2.1}$$

where γ_u and γ_v respectively denotes the u^{th} and v^{th} entry of γ.

Although there are infinitely many constructions that satisfy the above condition, we can derive a systematic method to derive kernel functions via procedurally combining simple *base* kernels to generate more sophisticated *composite* kernels. As formalized in Duvenaud et al. [15], a popular set of kernel composition rules can be described as follows:

- Let $\mathcal{M}$ be the set of all valid kernel functions, we first select $\mathcal{K}_{\mathcal{B}} \subset \mathcal{M}$ as a base kernel set. We say that $\mathcal{K}_{\mathcal{B}}$ induces a set of composite kernels $\mathcal{K}_{\mathcal{C}}$, such that $\mathcal{K}_{\mathcal{B}} \subset \mathcal{K}_{\mathcal{C}} \subseteq \mathcal{M}$. The membership of $\mathcal{K}_{\mathcal{C}}$ is recursively defined via the following rules:
- Addition: if $k_1, k_2 \in \mathcal{K}_{\mathcal{C}}$ and $\forall \mathbf{x}, \mathbf{x}' : k_+(\mathbf{x}, \mathbf{x}') \triangleq k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$, then $k_+ \in \mathcal{K}_{\mathcal{C}}$.
- Multiplication: if $k_1, k_2 \in \mathcal{K}_{\mathcal{C}}$ and $\forall \mathbf{x}, \mathbf{x}' : k_+(\mathbf{x}, \mathbf{x}') \triangleq k_1(\mathbf{x}, \mathbf{x}') \times k_2(\mathbf{x}, \mathbf{x}')$, then $k_\times \in \mathcal{K}_{\mathcal{C}}$.

For ease of notation, we will drop the input argument and instead treat kernel functions as basic operands (e.g., $k_+ = k_1 + k_2$ or $k_\times = k_1 \times k_2$) when referring to these composition rules. We further remark that the notation $k \in \mathcal{K}_{\mathcal{C}}$ only implies the *discrete form* of the kernel function, which does not include its learnable parameters. For example, consider the general squared exponential (SE) kernel $k_{\text{SE}} \in \mathcal{K}_B$ such that $k_{\text{SE}}(\mathbf{x}, \mathbf{x}') \triangleq \sigma^2 \exp\left(\sum_{t=1}^{d} \ell_t^{-2} \cdot (\mathbf{x}_t - \mathbf{x}'_t)^2\right)$ is parameterized by the signal variable σ and the length-scale variables $\boldsymbol{\ell} = \{\ell_t\}_{t=1}^{d}$. We do not count different initializations of k_{SE} in $\mathcal{K}_{\mathcal{C}}$ because the maximum likelihood estimation (MLE) of these parameters is well-studied and can be implicitly incorporated into the performance metric. Finally, the kernel selection problem can be formalized as follows:

Definition 3 (Kernel Selection) *Let $\tau \triangleq \{\mathcal{K}_{\mathcal{B}}, \mathcal{D}, \mathcal{A}\}$ describe a kernel selection task where $\mathcal{K}_{\mathcal{B}}$ is the set of base kernels; $\mathcal{D}$ is the set of provided observations (i.e., train/validation/test data); and $\mathcal{A}$ denotes a kernel-based algorithm specified by some $k \in \mathcal{K}_{\mathcal{C}}$ that is induced by $\mathcal{K}_{\mathcal{B}}$. We further let $F_\tau : \mathcal{K}_{\mathcal{C}} \rightarrow \mathbb{R}$ be the performance measuring function of this task, which (1) executes a well-defined algorithmic procedure to optimize the continuous parameters of some candidate kernel function k (e.g., applying* *gradient descent update with respect to the MLE objective until convergence) and (2) returns the predictive accuracy evaluated on the test set. Then, the kernel selection problem is succinctly stated as:*

$$k_* \;=\; \underset{k \in \mathcal{K}_C}{\operatorname{argmax}}\, F_\tau(k)\ , \tag{2.2}$$

where k_* denotes the optimally performing kernel function with respect to τ.

2.2 Related work

2.2.1 Heuristic tree search

Leveraging the tree structure induced by the composition rules above, Duvenaud et al. [15] approaches the KS problem by performing a heuristic tree search guided by the model likelihood score. Essentially, this heuristic approximates the performance measuring function in Definition 3 by fixing the kernel parameters during the tree search. The kernel parameters (of the selected base functions) are then trained after the tree search concludes and used as initialization for the next round of tree search. This procedure is then repeated until convergence or a sufficiently good kernel function is found. Nonetheless, the parameters optimized with respect to one kernel function might not induce similar predictive behaviors on another function, thus it is inconclusive whether the model likelihood heuristic accurately estimates the kernel performance.

2.2.2 Extending Bayesian optimization

Malkomes et al. [58] employs a special variant of the BO algorithm [75] in which the covariance function of the GP surrogate is specified by the Hellinger distance between the kernel-induced posterior distributions. While this method partially alleviates the initialization problem from Duvenaud et al. [15] by sampling kernel parameters from prior distributions to estimate the proposed Hellinger kernel, it cannot tractably update these priors as the BO algorithm acquires more observations. To address this problem, Lu et al. [57] first trains a variational autoencoder (VAE) model [45] to acquire a latent embedding space of kernel functions. Lu et al. [57] then employs BO to optimize for a latent representation that decodes into the optimal kernel function given the task data, thus bypassing the challenge of modelling a kernel function on discrete objects.

2.3 Motivation

We note that all of the above methods commonly require further restrictions of the candidate space to a finite set to ensure feasibility. Particularly, Duvenaud et al. [15] assumes a finite-depth tree such that the tree search algorithm can successfully terminate. The BO approach of Malkomes et al. [58] does not have a scalable solution for optimizing its acquisition function and has to confine the search space to a small number of active candidates, such that their acquisition values can be exhaustively computed. Similar to Duvenaud et al. [15], the VAE parameterization of Lu et al. [57] also assumes that the length of any decoded kernel expression is upper-bounded.

This section will now discuss a new approach to address the various shortcomings above through a more expressive reformulation that does not require any additional domain restriction. In particular, our framework, titled DTerGenS: Dynamic Termination Generative Search, casts the kernel selection problem as optimizing the weights of a kernel generative model, which explicitly mirrors the kernel composition rules via a recurrent formulation. As this weight domain is essentially a latent embedding space for kernel functions, our reformulation can be seen as taking a similar approach to Lu et al. [57], which alternatively employs a variational autoencoder [45] to construct this space.

Unlike Lu et al. [57], which treats each kernel expression as an atomic instance to be embedded on a continuous latent space, our approach instead maps an *entire subspace* of kernel expressions to each latent representation. Explicitly, this is achieved via formulating our generative model as an open-ended process which synthesizes an infinite trajectory of kernel expressions given a unique weight initialization. Our AAD objective can be seen as finding a subspace that contains the optimal model, rather than searching for the optimal model itself. This relaxation subsequently allows us to capture an unrestricted and significantly more expansive set of candidate kernel functions, compared to existing approaches such as Duvenaud et al. [15], Lu et al. [57], Malkomes et al. [58].

Nonetheless, infinitely complex kernel expressions are generally unmeaningful and impractical to compute. Given the result of our relaxed objective, the remaining technical challenge is therefore to distill the optimal kernel function from the optimal trajectory. We address this challenge by further introducing a data-driven policy that learns to optimally terminate the generative model to maximize performance. Intuitively, the interplay between our generator model and its termination policy can be interpreted as a two-step decision making process, where the former determines a subspace of candidates and the latter learns to select candidates in this generative trajectory. We then show that this policy can be jointly learned along with the generative model parameters using a bi-level Bayesian optimization scheme.

Last, we demonstrate that our method is able to produce complex kernels which significantly improve predictive performance of multiple predictive tasks over state-of-the-art structure search methods. Our results show a wider range of structures being explored by DTERGENS and more rapid rates of improvement as compared to other methods. Finally, we show that DTERGENS is also able to recover known well-performing kernels on artificially designed predictive tasks. The remainder of this chapter will now provide a summary of our technical approach, which will be presented in detail in Appendix A.

2.4 Technical Approach

Following the motivation above, we first state our representation of a kernel as the output of a generative process $G(\theta, \pi)$ parameterized by generative weights θ and termination policy π. Here, θ and π respectively determine an infinite trajectory of kernel expressions to be synthesized and the cut-off point to return a finite expression. We now reformulate the KS objective as:

$$\arg\max_{k \in \mathcal{K}_{\mathcal{C}}} F_\tau(k) \quad \simeq \quad \arg\max_{\theta \in \Theta, \pi \in \Pi} \Big[R_\tau(\theta, \pi) \ \triangleq \ (F_\tau \circ G)(\theta, \pi) \Big] , \tag{2.3}$$

where Θ and Π denote the space of generative weights and termination policies; τ denotes the kernel selection task defined above; and R_τ conveniently denotes the composite function $F_\tau \circ G$. We note that a composite kernel expression can be written as a sum-of-products over base kernel units. That is, for any composite kernel $k \in \mathcal{K}_{\mathcal{C}}$, there exists a finite collection of base kernels $k_{t,t'} \in \mathcal{K}_{\mathcal{B}}$ such that:

$$k \ = \ \sum_{t=1}^{m} \prod_{t'=1}^{n_t} k_{t,t'} \ . \tag{2.4}$$

Any composite kernel expression can be viewed as a sub-tree in this search space defined by a *primary chain* (i.e., the summation over product terms) connecting multiple *secondary branches* (i.e., products of base kernel units). To generate such structures, we construct our generator G by composing two nested recurrent neural networks, which alternately expand a candidate functional expression via (1) generating a new secondary branch or (2) generating a new base kernel function to an existing branch. An overview of our generator is given in Fig 2.1. We give the full details of our proposed generator in Appendix A

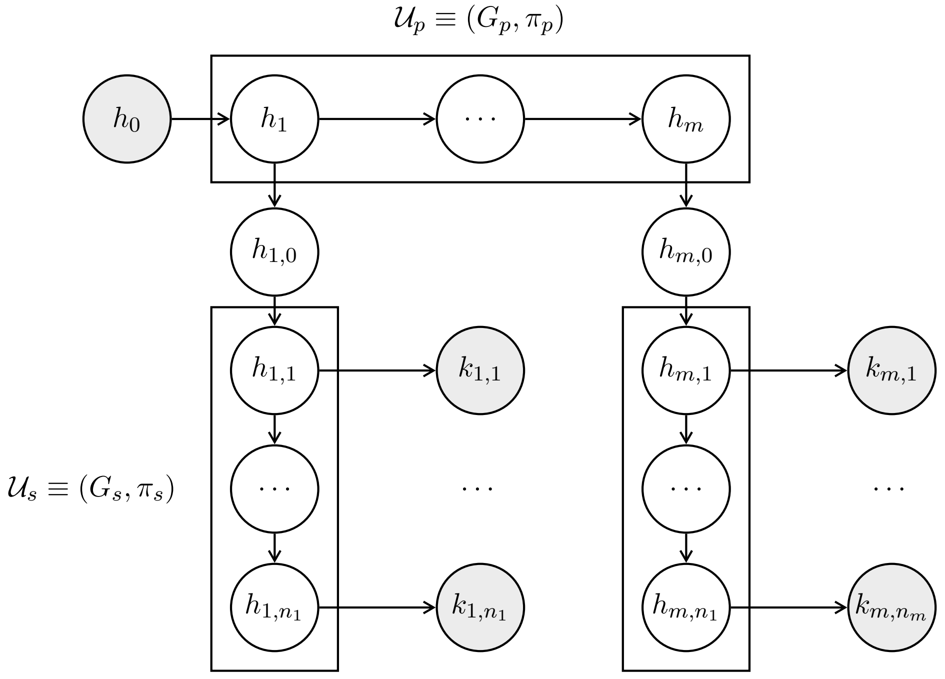


Figure 2.1: Schematic of the kernel generator with nested units $\mathcal{U}_p$ and $\mathcal{U}_s$. Each component recursively computes its next hidden state and emission output using respective recurrent neural network G_s and G_p. The termination probability at each generative step is determined by policies π_p and π_s. The final candidate kernel expression is composed using Eq. (2.4).

Our termination policy π is then given by a pair of functions that correspond to these component networks and are modelled by simple feed-forward neural networks. Specifically, each function takes as input an intermediate kernel expression and estimates the probability that its respective component network should be terminated to yield the optimal kernel function.

We propose a bi-level optimization scheme to sequentially optimize both these components. At every update iteration, we use the Bayesian optimization (BO) framework to obtain and evaluate a candidate generative weights θ. Fixing a policy π allows us to distill an concrete kernel expression from the induced generative trajectory and consequently obtain its respective performance. The partial trajectory obtained is then used to update the policy networks. A high-level overview of this algorithm is given in Fig. 2.2.

Although the use of BO is similar to Lu et al. [57] and Malkomes et al. [58], specific considerations need to be applied to account for the differences in settings between our proposal and these approaches.

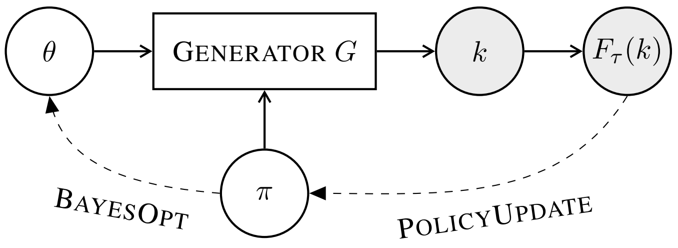


Figure 2.2: The generic workflow of DTERGENS. Given policy π, we employ BO to obtain generative weight candidate θ (Section A.1.3). Using the observed generative trajectory, we alternately update the policy distribution (Section A.1.4).

For instance, a major technical challenge that arises in our formulation is that the semantics of our latent representations will dynamically change between BO iterations as the policy π is updated. Explicitly, even though each θ encodes a unique generative trajectory, the performance obtained by querying R_τ also depends on the early stop decisions drawn from the current policy π. As such, it is necessary that our surrogate model can accurately capture the correlation between different instantiations of π.

To achieve this, we first impose Gaussian priors on the weights of our policy networks and further assume that termination policy π is constructed at every iteration via sampling from the respective weight distributions. As each policy is now implicitly represented by a pair of distributions, we subsequently propose to estimate the correlation between two instantiations of π via computing their corresponding Jensen-Shannon divergences, which are then incorporated into the BO surrogate model. Finally, given an observed (partial) trajectory at the end of each BO iteration, we further devise an update strategy for the weight distributions of π via minimizing their expected deviation from the actual optimal stopping point. The full detail of our technical contributions and empirical demonstrations is given in Appendix A.

2.5 Results

To demonstrate the performance of DTERGENS, we compare our method with the following benchmarks: (a) random search over the space of kernels with max length $L \leq 10$ (baseline); (b) SVO: Structure Variationally-Encoded Optimization [57], for which we train the VAE component using 25000 randomly generated kernel expressions with max length $L \leq 10$ (to show the advantage of generative search); and

(c) our own algorithm with no stopping policy and fixing expression length $L = 2, 4, 8$. For (c), the termination of the secondary component is chosen at random, the termination of primary component is guaranteed upon reaching length L, and REMBO [79] is used to optimize generative weights θ. These benchmarks serve as ablation studies to demonstrate the advantage of having adaptive termination policies for the generative components.

We first demonstrate that our kernel selection framework can accurately recover a synthetic kernel structure given observations simulated by its corresponding distribution. In particular, we arbitrarily construct several kernels with different complexities and show that in all scenarios, our method DTERGENS outperformed other benchmarks in terms of recovery error (i.e., the Frobenius-norm of the distance between ground truth annd reconstructed covariance matrices).

We further demonstrate the effectiveness of our approach on several real-world regression datasets, including: (1) the DIABETES dataset [17], which contains 442 diabetes patient records; (2) the MAUNA LOA dataset [44], which measures monthly average CO_2 concentration at the Mauna-Loa Observatory over 42 years; and (3) the PROTEIN dataset [69], which features 45730 protein tertiary structures. Our empirical results show that DTERGENS outperformed other benchmarks in terms of prediction errors, the implying the discovery of better kernel structures.

The full detail of our experiments are given in Appendix A. This work has been published at the International Conference on Machine Learning (2020).

Chapter 3

Minimizer Sketch Design

3.1 Problem setting

The minimizer scheme [72, 73] is a sampling method that selects length-k substrings (i.e., k-mers) from a string such that sufficient information about the identity of the sequence is preserved, typically for comparison purpose. It is widely used to reduce memory consumption and run-time in bioinformatics applications such as genome assemblers [82], read mappers [41, 51] and k-mer counters [14, 19].

Our discussion of the minimizer scheme will be grounded in sequences drawn from an arbitrary alphabet Σ. To begin, let us define some useful notations. Given some parameter k, a (w, k)-window is defined as a substring of length $w_k = w - k + 1$, which contains exactly w overlapping k-mers. Let $S \in \Sigma^{L+k-1}$ be sa string containing exactly L overlapping k-mers and $L_w \triangleq L - w + 1$ overlapping (w, k)-windows. Generally, we assume that $w \ll L$ and $k \ll L$, as is typical in most practical settings. We use the notations κ_i^k and $\kappa_i^{w_k}$ to respectively denote the i^{th} k-mer and the i^{th} (w, k)-window in S.

Generally, a k-mer sampling scheme is a function $\mathcal{X}$ such that $\mathcal{X}(S) \in 2^{[L]}$ returns a set of k-mer locations in S. The resulting sketch is given by the notation $\mathcal{K}(S; \mathcal{X}) \triangleq \{(\kappa_i^k, i)\}_{i \in \mathcal{X}(S)}$. Note that the sketch report tuples of both k-mer and index to distinguish identical k-mers sampled from different parts of S. We evaluate the performance of such a sampling scheme by computing its density [59], which is the fraction of sampled k-mers relative to the length of the target sequence:

$$D(S; \mathcal{X}) \triangleq \frac{|\mathcal{K}(S; \mathcal{X}, r)|}{L} . \tag{3.1}$$

The minimizer scheme [73] is a k-mer sampling scheme that makes its sampling decisions based on an arbitrarily constructed total ordering on the k-mer set, which is represented by a permutation π. In particular, we define the minimizer scheme as follows:

Definition 4 (Minimizer) *A minimizer scheme is characterized by a tuple of parameters (w, k, π), where w, k are defined above and π is a total ordering on the set of all k-mers. We define the following selector function, which returns the lowest-ranked k-mer in some (w, k)-window of S:*

$$m(\kappa_v^{w_k}; \pi) \triangleq \underset{i \in [1,w]}{\operatorname{argmin}} \sum_{j \in [1,w]} \mathbb{I}(\kappa_{v+j-1}^k <_\pi \kappa_{v+i-1}^k) , \tag{3.2}$$

where κ_{v+i-1}^k denotes the i^{th} k-mer in the window $\kappa_v^{w_k}$ and $\mathbb{I}(\kappa <_\pi \kappa')$ denotes the event that κ precedes κ' in π for some $\kappa, \kappa' \in \Sigma^k$. The minimizer sampling function is then given by:

$$\mathcal{M}(S; w, k, \pi) \triangleq \{i + m(\kappa_i^{w_k}; \pi)\}_{i \in [L_w]} , \tag{3.3}$$

which iteratively applies m on every window in S.

A low-density minimizer scheme achieves three desiderata: (1) the sketch is compact and will offer significant cost saving to downstream applications; (2) every (w, k)-window in S overlaps at least one k-mer in the sketch; and (3) identical windows are represented by the same k-mer due to the deterministic sampling protocol. These properties give rise the minimizer sketch design (MSD) problem below.

Definition 5 (Low Density Minimizer Sketch Design) *Let S be a string defined as above. Suppose w, k and r are given as application-specific parameters, the minimizer selection problem is defined as:*

$$\pi_* = \underset{\pi \in \Pi(\Sigma^k)}{\operatorname{argmin}} \mathcal{D}(S; \mathcal{M}(\pi)) , \tag{3.4}$$

where $\Pi(\Sigma^k)$ denotes the set of all k-mer permutations in Σ^k and we write $\mathcal{M}(\pi)$ to clearly show the dependency of $\mathcal{M}$ on the k-mer ordering π, which is being optimized.

Another recently suggested sketching desiderata [16] is that two substrings differing only by a few mutations are likely represented by the same k-mer in the sketch. Edgar [16] argues that the density metric above is not sufficient to capture this, and hence proposes an alternative metric called *conservation*:

$$C(S; \mathcal{X}) \triangleq \mathbb{E}_{S'} \left[\frac{|\mathcal{K}(S; \mathcal{X}) \cap \mathcal{K}(S'; \mathcal{X})|}{L} \right] , \tag{3.5}$$

which measures the expected fraction of selected k-mers (relative to the length of S) that are preserved under random mutation (i.e., higher is better). Here, S' is a copy of S with randomly substituted characters and the conservation metric is expected over the distribution of S'. To obtain high conservation sketches Edgar [16] also proposes another sketching method called *syncmers*.

Definition 6 (Syncmer) *A syncmer scheme is defined by a tuple of parameters (k, s, t, π), where k similarly denotes the substring length to be sampled (i.e., k-mers) and $s, t \in [1, k]$. Here, π is a permutation of all s-mers in Σ^s. Similar to Definition 4, we also define a selector function, which now returns the lowest-ranked s-mer in some k-mer of S:*

$$m(\kappa_v^k; \pi) \quad \triangleq \quad \arg\min_{i \in [1,k]} \sum_{j \in [1,k_s]} \mathbb{I}(\kappa_{v+j-1}^s <_\pi \kappa_{v+i-1}^s) \ , \tag{3.6}$$

where $k_s \triangleq k - s + 1$ denotes the number of overlapping s-mer in a k-mer. The syncmer sampling function is then specified by iteratively sampling any k-mer such that its lowest-ranked s-mer is at the t^{th} position:

$$\mathcal{O}_t(S; k, s, \pi) \triangleq \{i \mid m(\kappa_i^k; \pi) = t\}_{i \in [L]} \ . \tag{3.7}$$

We specially choose to write t as a subscript since syncmer schemes with similar k, s, π initialization and different offset t are theoretically related, as shown below in the technical approach.

Although there has been no explicit sketch design method developed for syncmers (i.e., optimizing the s-mer permutation π), Edgar [16] and Shaw and Yu [74] have empirically observed that the syncmer method is capable of simultaneously achieving both lower density and higher conservation than the minimizer method when both use a random ordering. Nonetheless, no formal relationship has been established between the two methods due to the lack of a convention to compare them.

This chapter explores a new approach for solving the minimizer sketch design problem via a differentiable reformulation of the permutation learning task in Definition 4, called DEEPMINIMIZER. We then derive the first theoretical relationship between the density and conservation metrics; as well as between the minimizer and syncmer methods. This new insight allows us to (1) unify minimizers and syncmers under a generalized notion of sequence sketching called *masked minimizer*; and (2) extend our DEEPMINIMIZER framework to automate *masked minimizer* sketch design.

3.2 Related work

3.2.1 UHS-based methods

Most existing minimizer selection schemes with performance guarantees over random sequences are based on the theory of universal hitting sets (UHS) [60, 67]. Particularly, a (w, k)-UHS is defined as a set of k-mers such that every window of length w (from any possible sequence) contains at least one of its elements. Every UHS subsequently defines a family of corresponding minimizer schemes whose expected densities on random sequences can be upper-bounded in terms of the UHS size [59]. As such, to obtain minimizers with provably low density, it suffices to construct small UHS, which is the common objective of many existing approaches [18, 59, 85]. These methods, however, rely on the unrealistic assumption that the target sequences follow a uniform distribution [84]. As such, there tends to be little correspondence between the provable upper-bound on expected density and the actual density measured on a target sequence.

3.2.2 Heuristic methods

Several minimizer construction schemes rank k-mers based on their frequencies in the target sequence [11, 40], such that infrequent k-mers are more likely to be chosen as minimizers. These constructions nonetheless rely on the assumption that infrequent k-mers are spread apart and ideally correspond to a sparse sampling. Another greedy approach is to sequentially remove k-mers from an arbitrarily constructed UHS, as long as the resulting set still hits every w-long window on the target sequence [12]. Though this helps to fine-tune a given UHS with respect to the sequence of interest, there is no guarantee that such an initial set will yield the optimal solution after pruning.

3.2.3 Polar set construction

Recently, a novel class of minimizer constructions was proposed based on polar sets of k-mers, whose elements are sufficiently far apart on the target sequence [86]. The sketch size induced by such a polar set is shown to be tightly bounded with respect to its cardinality. This reveals an alternate route to low-density minimizer schemes through searching for the minimal polar set. Unfortunately, this proxy objective is NPhard and currently approximated by a greedy construction [86], which can be sub-optimal in practice.

3.3 Differentiable reformulation of the MSD problem

3.3.1 Motivation

The technical challenges of the MSD problem described above arise due to two factors. First, the search space is factorially large in terms of k. Exhaustively searching this domain of k-mer permutations would suffice for very small k, but will quickly become intractable for larger values of k that are typically used in many practical applications. Second, the density minimizing objective is discrete, hence difficult to be optimized via standard techniques.

All existing MSD frameworks approach this optimization problem by approximating its permutation search space (i.e., space of total k-mer orderings) with the space of partial k-mer orderings that satisfy certain surrogate properties. For example, Ekim et al. [18], Marçais et al. [59, 60], Zheng et al. [85] adopt the universal hitting set (UHS) approximation which imposes that all k-mers in a selected UHS will be ranked with lower priorities than those outside. The polar set (PS) approximation proposed by Zheng et al. [86] is similar in spirit, but uses a different set of surrogate properties to construct these partial orderings.

The main advantage of these strategies is that they reduce the permutation learning objective in Definition 5 to finding the most compact construction of such surrogate sets, which is relatively simpler to achieve. However, these approximations either rely on unrealistic assumptions about the target sequence, such as its characters are uniformly distributed (e.g., UHS-based methods [18, 59, 60, 85]); or remain a challenging discrete objective that can only be solved via greedy heuristics (e.g., the PS method [86]). Furthermore, both of these approximation schemes can be viewed as domain restriction techniques where the sets of active candidates are confined to UHS/PS-induced permutations. Nonetheless, there is no guarantee that these active sets would necessarily contain the optimal solution.

To overcome these challenges, we instead propose a re-parameterization of the original permutation learning problem, which implicitly casts the k-mer permutation π as a function $f_\pi : \Sigma^k \rightarrow [0, 1]$ that assign continuous scores to k-mers in Σ^k. Every *valid* candidate function must be a consistent scoring scheme (i.e., a k-mer will get the same score regardless of its position and local neighborhood in the target sequence), such that a permutation can always be recovered via sorting the scores. Modelling the space of such valid functions with a deep neural network, we subsequently propose to cast the MSD problem as a deep learning optimization task with respect to the density objective in Definition 5. We further note that, unlike existing approximations, the candidate set encoded by our re-parameterization can theoretically approach the unrestricted permutation domain $\Pi(\Sigma^k)$ given a sufficiently expressive network architecture.

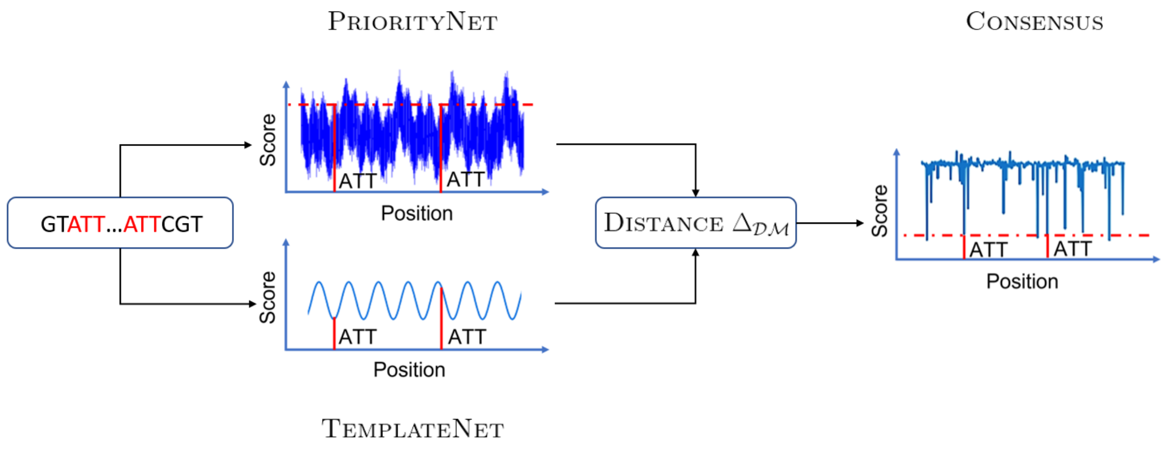


Figure 3.1: The DEEPMINIMIZER framework is comprised of two continuous k-mer scoring schemes that correspond to different aspects of a low-density minimizer: PRIORITYNET assigns scores to k-mers in the target sequence such that a total ordering can be recovered via simple sorting; whereas TEMPLATENET relaxes this consistency constraint to achieve a low-density sketch. Minimizing the distance between their outputs is expected to yield a consensus solution that simultaneously exhibits both properties.

Nonetheless, standard backpropagation methods [46] cannot be directly applied to optimize the density metric, which is discrete and does not have an analytic gradient with respect to the network parameters. To address this challenge, we further adopt a similar approach to the proposed KS framework in Chapter 2, which hierarchically approximates the search task with multiple sub-tasks that can each be solved more efficiently. In the MSD context, these sub-tasks manifest as a pair of complementary neural networks called PRIORITYNET and TEMPLATENET. In particular, the PRIORITYNET component is parameterized such that it always outputs consistent scoring and is tasked to encode the candidate function space of the minimizer scheme. On the other hand, the TEMPLATENET component relaxes this consistency constraint in exchange for the ability to encode desirable score assignments (but potentially unmeaningful since it might not be possible to recover a proper minimizer scheme). While the KS framework in Chapter 2 adopts a bi-level optimization approach to alternately solve its sub-tasks (i.e., generative trajectory selection and early stopping policy), both components in this MSD scenario can be simultaneously learned via minimizing the divergence between their outputs, hence resulting in the first differentiable relaxation for the MSD problem. A high-level schematic of our approach is shown in Fig. 3.1. Last, we propose to demonstrate that our method, titled DEEPMINIMIZER is highly-efficient and capable of finding minimizer schemes that yield significantly more compact sketches on multiple human genome benchmarks. The remainder of this chapter will now provide a summary of our technical approach, which will be presented in details in Appendix C along with our empirical findings.

3.3.2 Technical Approach

Using the notion of the scoring function f_π, the selector function in Definition 4 can be written as:

$$m(\kappa_v^{w_k};\alpha) \;\triangleq\; \underset{i\in[1,w]}{\arg\min}\, f_\pi\left(\kappa_{v+i-1}^{k};\alpha\right)\ , \tag{3.8}$$

where $f : \Sigma^k \rightarrow [0,1]$ is parameterized by a convolutional neural network with weight α. The architecture of this network (as discussed in Appendix C) guarantees that f is a valid scoring function (i.e., we can always recover a total ordering over all k-mers), that is, for all pairs of indices $i, j \in [L]$, we have:

$$\kappa_i^k = \kappa_j^k \quad \Longrightarrow \quad f(\kappa_i^k) = f(\kappa_j^k)\ . \tag{3.9}$$

For clarity, we will write the density metric as $\mathcal{D}(S;\mathcal{M}(\alpha))$ to clearly establish the change in parameterization. As discussed in the previous section, f cannot be efficiently optimized with gradient backpropagation methods since the derivative $\frac{\partial\mathcal{D}(S;\mathcal{M}(\alpha))}{\partial\alpha}$ does not exist. To work around this, we introduce a proxy optimization objective that approximates $\mathcal{D}(S;\mathcal{M}(\alpha))$ via coupling f with a positional scoring function $g : [L] \rightarrow [0,1]$ parameterized by weight β. Unlike f, which assigns k-mers scores based on their contents, g assigns k-mers scores based on their positions in S and is likely not a valid scoring function, i.e., $\kappa_i^k = \kappa_j^k \;\not\Longrightarrow\; g(i) = g(j)$. Nonetheless, the context-free function g can be used to generate a k-mer sketch by adapting Eq. 3.3 as:

$$\mathcal{T}(S;\beta) \triangleq \left\{ i + \underset{j\in[1,w]}{\arg\min}\, g(i+j-1;\beta) \right\}_{i\in[L_w]}\ . \tag{3.10}$$

The *template* k-mer sampling scheme $\mathcal{T}$ does not have the same desiderata as a minimizer sketch, but we can guarantee that the sketch density is equal to that of an optimal minimizer sketch through carefully constructing g, i.e., $\mathcal{D}(S;\mathcal{T}(\beta)) \simeq 1/w$. As $\mathcal{K}(S;\mathcal{M}(\alpha),r)$ corresponds exactly to a minimizer sketch and $\mathcal{K}(S;\mathcal{T}(\beta),r)$ approximates the low-density objective, this reveals an interesting factorization of the search task, and therefore an alternative pathway to the optimal solution through finding a *consensus solution* between the two sketches. Formally, let $\mathbf{f}(S,\alpha) = [f(\kappa_i^k;\alpha)]_{i\in[L]}$ and $\mathbf{g}(\beta) = [g(i)]_{i\in[L]}$ respectively denote the concatenated score vectors that f and g assign to k-mers in S, we then characterize this consensus solution as a joint instantiation of α and β such that some distance metric Δ between $\mathbf{f}(S,\alpha)$ and $\mathbf{g}(\beta)$ is minimized. Explicitly, this results in the following optimization objective:

$$(\alpha_*,\beta_*) \triangleq \underset{\alpha,\beta}{\arg\min}\, \Delta\left(\mathbf{f}(S,\alpha),\mathbf{g}(\beta)\right) \ , \tag{3.11}$$

which is fully differentiable with respect to weights α and β. We fully describe our approach, including the parameterizations of f, g and the distance metric Δ in Appendix C.

3.4 Unifying minimizer and syncmer sketching methods

3.4.1 Motivation

Although both minimizers and syncmers employ the same concept of sampling based on a substring total ordering, a minimizer scheme that uses an s-mer ordering will directly report s-mers in its sketch, whereas a syncmer scheme parameterized by the same ordering will typically report longer k-mers. Due to this mismatch in representation, existing work [16, 74] all chose to compare schemes that report similar-sized substrings regardless of their ordering choices, which results in an asymmetry of information among their sketches and prevents the derivation of any meaningful correspondence between minimizers and syncmers.

This shortcoming motivates a revision of the comparability notion for sketching methods. In particular, we advocate comparing schemes that use the same substring ordering to make sampling decisions. This new mode of comparison allows us to explicitly bound the density and conservation gaps between minimizers and syncmers. Building on this theoretical result, we further propose a novel concept of masked minimizers that unify both minimizers and syncmers. The masked minimizer scheme combines the standard minimizer sampling with an additional sub-sampling step, which applies a binary mask filter to the k-mer selection at every window. Interestingly, varying this mask parameter induces a spectrum of comparable schemes and reveals a methodical approach to derive comparable sketching schemes. Last, we propose an extension of the DEEPMINIMIZER method to optimize the masked minimizer scheme with respect to a novel sketching metric called the *generalized sketch score* (GSS), which combines both density and conservation metrics, resulting in the first formal protocol to compare sequence sketching methods.

3.4.2 Technical approach

To explicitly reason about the difference in terms of sketch representation between minimizers and syncmers, we introduce the notion of a reporting function r, which maps the sampled locations in $\mathcal{X}(S)$ to tuples of substrings and locations. That is, the reported sketch is now constructed as $\mathcal{K}(S; \mathcal{X}, r) \triangleq \{r(i)\}_{i \in \mathcal{X}(S)}$. Setting $r(i) = (\kappa_i^k, i)$ recovers both Definition 4 of a minimizer scheme with parameters w, k and $\pi_m \in \Pi(\Sigma^k)$; and Definition 6 of a syncmer scheme with parameters k, s, t and $\pi_o \in \Pi(\Sigma^s)$. Since these methods both report k-mers, they are traditionally deemed comparable.

This mode of comparison, however, does not facilitate a theoretical analysis of their performance differences (i.e., in terms of density and conservation metrics), as different information bases are used to enact the sampling decisions of a length-k minimizer and syncmer. In particular, (w, k)-minimizer schemes use total k-mer orderings to perform sampling, whereas (k, s, t)-syncmer schemes use total s-mer orderings, with $s \leq k$. We note that these bases are only comparable when setting $s = k$, but doing so results in trivial syncmer schemes that selects every k-mer in S.

To correct this asymmetry of information, we propose the notion of π-comparable minimizers and syncmers, which are schemes that employ the same ordering π. For example, the (w, k, π)-minimizer and the (w_k, k, t, π)-syncmer schemes with $w_k \triangleq w + k - 1$ and any $t \leq w$, which respectively report k-mers and w_k-mers, are π-comparable. To normalize the difference in representation, we replace the default w_k-mer reporting function $r(i) = (\kappa_i^{w_k}, i)$ of the above syncmer scheme with the k-syncmer reporting function $r'(i) = (\kappa_{i+t}^k, i + t)$. As both reporting functions are one-to-one mappings of the selected locations, this substitution results in a semantically equivalent *shifted* syncmer scheme.

This translation of the reporting function aligns our proposed comparison with the traditional perspective of comparability and presents an invariant substring ranking behavior among the compared schemes, which subsequently allows us to derive explicit bounds on their density and conservation gaps. In particular, we prove that the density and conservation of any syncmer on a specific sequence S are respectively upper-bounded and almost surely upper-bounded by that of its π-comparable minimizer, thus establishing the first theoretical correspondence between π-comparable schemes. Our theoretical result is summarized in Theorem 3.4.1, whose detailed proof is given Appendix C.

Theorem 3.4.1 (π-comparability defines a correspondence between minimizers and syncmers) *For every syncmer scheme $(\mathcal{O}_t, r')$, there exists a comparable minimizer scheme $\mathcal{M}$ whose reporting function r is translated from r' with an offset t, such that the following bounds hold for every $S \in \Sigma^{L+k-1}$:*

$$D(S; \mathcal{O}_t, r') \;\leq\; D(S; \mathcal{M}, r) \qquad \text{and} \qquad C(S; \mathcal{O}_t, r') \;\leq\; C(S; \mathcal{M}, r) + \frac{t}{L} \,. \tag{3.12}$$

We then further show that any syncmer scheme can be recovered given a π-comparable minimizer $\mathcal{M}$ via sub-sampling the set of locations selected by $\mathcal{M}$. We write this sub-sampling rule as follows:

$$\mathcal{O}_t(S) \;\;=\;\; \left\{ i - t \mid \left\| \overrightarrow{m}(\kappa^{w_k}_{i-t}; \pi) \otimes \mathbf{e}_t \right\|_1 > 0 \right\}_{i \in \mathcal{M}(S)} \,, \tag{3.13}$$

where $\overrightarrow{m}(\kappa^{w_k}; \pi) \triangleq [\mathbb{I}(j = m(\kappa^{w_k}; \pi))]_{j \in [w]}$ is the one-hot representation of the k-mer selection result at some arbitrary window κ^{w_k} using the selector function m; $\mathbf{e}_t$ denotes the one-hot vector with a non-zero entry on the t^{th} row; and $\otimes$ denotes the pointwise multiplication operator. The sub-sampling condition above checks if any position remains selected after filtering with $\mathbf{e}_t$ (in which case, it must be the t^{th} position). Interestingly, this sub-sampling rule can be generalized by replacing $\mathbf{e}_t$ with any arbitrary w-dimensional binary mask $\nu \in \{0, 1\}^w$. For example, setting $\nu = \mathbf{1}_w$ trivially recovers the standard minimizer scheme (without applying the offset t). Given a set of minimizer-sampled locations $\mathcal{M}(S)$, varying ν yields a total of 2^w π-comparable schemes, leading to a unifying method called *masked minimizers*.

Definition 7 (Masked minimizers) *The sampling function of a masked minimizer scheme is characterized by a tuple of parameters (w, k, π, ν), where w, k, π correspond to standard minimizer parameters, and $\nu \in \{0, 1\}^w$ is a w-dimensional binary vector. The masked minimizer sampling function is given by:*

$$\mathcal{V}(S; w, k, \pi, \nu) \;\triangleq\; \{ i + m(\kappa^{w_k}_i; \pi) \mid \zeta(\kappa^{w_k}_i, \nu) \}_{i \in [L_w]} \,, \tag{3.14}$$

where $\zeta(\kappa^{w_k}_i, \nu) \triangleq \|\overrightarrow{m}_\pi(\kappa^{w_k}_i) \otimes \nu]\|_1 > 0$ denotes the event that the selection at the i^{th} window remains sampled after applying the sub-sampling mask.

We also show that improving density places an upper-bound on how much conservation can be improved and vice versa, which implies that neither objective should be considered independently of one another. To account for this trade-off, we propose a new metric called the *generalized sketch score* (GSS):

$$G(S;\mathcal{X},r,w) \triangleq \frac{C(S;\mathcal{X},r)}{D(S;\mathcal{X},r)} \cdot \frac{1}{L_w}\sum_{i=1}^{L_w} V_i(S;\mathcal{X},w) \,, \tag{3.15}$$

where $V_i(S;\mathcal{X},w) \triangleq 1 - \prod_{j=i}^{i+w-1} \mathbb{I}(j \notin \mathcal{X}(S))$ is the indicator variable of the event that the window $\kappa_i^{w_k}$ overlaps at least one sampled location in $\mathcal{X}(S)$. The GSS metric seeks to capture the trade-off between density and conservation via the ratio $C(S;\mathcal{X},r)/D(S;\mathcal{X},r)$ and further prevents possible trivial exploitation with the *coverage* term $\sum_{i=1}^{L_w} V_i(S;\mathcal{X},w)$. We provide the full explanation regarding the benefits of this metric in Appendix C.

Finally, we propose an extension of the DEEPMINIMIZER method to optimize a masked minimizer scheme with respect to the GSS metric. In particular, our new objective is given as a bilvel optimization:

$$\begin{aligned}(\alpha_*,\beta_*) &\triangleq \arg\min_{\alpha,\beta} \Delta\left(\mathbf{f}(S,\alpha),\mathbf{g}(\beta)\right) + \sum_{i=1}^{n} \Delta\left(\mathbf{f}(S_i,\alpha),\mathbf{f}(S,\alpha)\right) \,, \\ \nu_* &\triangleq \arg\min_{\nu} G(S;\mathcal{V}(\alpha_*),r,w) \,, \end{aligned} \tag{3.16}$$

where $S_1, S_2, \ldots, S_n$ denote n randomly sampled mutations of S; Δ, $\mathbf{f}$ and $\mathbf{g}$ are previously defined in the technical exposition of the DEEPMINIMIZER method. The first optimization adds an extra term to the DEEPMINIMIZER loss, which measures the expected distance between each priority vector $\mathbf{f}(S_i,\alpha)$ for mutated sequence S_i and the original template vector $\mathbf{f}(S,\alpha)$. As the priority vector is a surrogate for the actual discrete k-mer selection, minimizing this term will likely improve the stability of selection (i.e., conservation) when the sequence is subjected to random mutations. The second optimization, on the other hand, is written as a greedy search to find the optimal mask given an ordering induced by the parameter α_* obtained from the first optimization. We detail our approach in Appendix C.

3.5 Results

To demonstrate the performance of DEEPMINIMIZER, we compare our method with the following benchmarks: (a) A random minimizer (the total ordering π is uniformly sampled); (b) MINICEPTION [85]; (c) PASHA [18]; and (d) POLARSET [86]. We use the following sequence benchmarks to conduct our empirical study: (a) human chromosome 1 (CHR1); (b) human chromosome X (CHRX); (c) the centromere region of chromosome X (CHRXC); (d) the full human genome (HG38). The full details of these sequences are given in Appendix C.

We first compare the minimizer sketch before and after training with DEEPMINIMIZER to demonstrate that DEEPMINIMIZER results in a sparse selection of k-mers in the target sequence. Across many settings of w and k, we show that DEEPMINIMIZER consistently converges to a low-density sketch, which shows that our objective function correlates well with the density metric. We also conduct ablation studies to study the effect of different template functions and distance functions, which justify our empirical choices.

To demonstrate the effectiveness of our masked minimizers unification approach and justify our theoretical insight, we further conduct extra experiments to answer the following questions: (1) Are density and conservation adversarially related? (2) How do π-comparable schemes perform relative to one another under the proposed metric GSS? (3) Can mask optimization improve the overall performance of both minimizer and syncmer? Through these demonstrations, we confirm our theoretical understanding of various sketching metrics and the relationship among π-comparable schemes, as well as demonstrate the efficiency of our proposed optimization method.

Our experiments show that density and conservation metrics indeed have an adversarial relationship. In addition, our masked minimizer training method is highly effective in optimizing their trade-off. We demonstrate this effectiveness in many settings of w, k and binary mask ν. Last, we also demonstrate that selecting an optimal mask can have significant effect on the GSS of sketching methods. We test this on sequence patterns that are known to be difficult for standard minimizer methods and show that our approach are capable of finding masks that improve the performance.

We give the full details of our empirical study to Appendix C. Our work on the DEEPMINIMIZER method has been published at the International Conference on Research in Computational Molecular Biology (RECOMB 2022). Our work on the unifying masked minimizer approach is currently under review for RECOMB 2023.

Chapter 4

Neural Architecture Search

4.1 Problem setting

Neural network architecture is the core element of deep learning's success on many perceptual tasks such as computer vision [50] and natural language processing [78]. Even though most renowned architectures in deep learning literature are hand-designed by domain experts, recent studies [35, 81, 87] have suggested that searching for an optimal design that composes well-known building blocks can significantly improve performances in many task domains. Formally, a neural network architecture can be written as a hierarchical feature extractor which explicitly takes the form of a directed acyclic graph $G \triangleq (V, E)$, where V and E respectively denote the sets of nodes and edges in G. Each node $v \in V$ denotes an intermediate feature representation z_v, which has been arbitrarily transformed from some input $\mathbf{x} \in \mathbb{R}^d$. We often associate $\mathbf{x}$ and the output of the network with a source node v_s and a sink node v_t, respectively. Each edge $e \equiv (v, v') \in E$ encodes an operator o_e that transforms z_v and forwards the result to v' to be aggregated as $z_{v'}$. The computation of any intermediate representation in G can then be recursively defined as:

$$z_{v'} \triangleq \sum_{(v,v') \in E} o_{(v,v')}(z_v) \ , \tag{4.1}$$

where we have assumed a simple additive aggregation rule for simplicity.

In a typical neural architecture search task, there are two design choices to be made: (1) the flow of information determined by the topology of G; and (2) the corresponding transformations of data features encoded by the edge operators o_e for every $e \in E$. To ensure feasibility, the space of edge operators is often restricted to a small, finite set $\mathcal{O}$ of well-known layers, such as linear transformation, convolution and pooling, thus reducing the problem to finding an optimal mapping $m : E \rightarrow \mathcal{O}$ that assigns an operator in $\mathcal{O}$ for every edge. Similar to the setting of the kernel selection problem, we are also interested in only representing the *categorical types* of these layers in $\mathcal{O}$, which do not include specific parameters whose optimization is well studied (i.e., layer weights) or must be chosen to ensure the mathematical consistency of the network (i.e., layer dimensions). In particular, given a selected layer type, the learning/setting of these parameters is implicitly incorporated into the performance measuring function.

On the other hand, designing the search space for G is still an active research direction in which no conclusive insight has been reached regarding the best instantiation. This thesis thus focuses on a common practice proposed by Bender et al. [6], which uses an over-parameterized graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to implicitly define this space. That is, a candidate graph $G = (V, E)$ can be obtained by selecting subsets of nodes $V \subseteq \mathcal{V}$ and edges $E \subseteq \mathcal{E}$ such that G is connected and contains both the source node v_s and sink node v_t. We note that this pruning step can be trivially modelled by including a null layer in $\mathcal{O}$, which outputs zero regardless of the input feature, such that whenever this layer is selected, the edge it represents is considered inactive in the candidate graph. Finally, the neural architecture search problem on an overparameterized search space can be described as follows:

Definition 8 (Neural Architecture Search) *Let $\tau \triangleq \{\mathcal{G}, \mathcal{O}, \mathcal{D}\}$ describe a neural architecture search task where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denotes an over-parameterized computation graph; $\mathcal{O}$ denotes the set of all edge operators; and $\mathcal{D}$ is the provided observations (i.e., train/validation/test data). We further let $\mathcal{M}$ be the set of all mappings $m : \mathcal{E} \rightarrow \mathcal{O}$ that assign a feature mapper for every edge in $\mathcal{G}$, thus inducing a unique architecture. Finally, let $F_\tau : \mathcal{M} \rightarrow \mathbb{R}$ be the performance measuring function of this task, which (1) executes a well-defined algorithmic procedure to optimize the parameters of the selected operators (e.g., applying gradient descent update with respect to the cross-entropy loss until convergence) and (2) returns the predictive accuracy evaluated on the test set. Then, the neural architecture search problem is succinctly stated as:*

$$m_* \;=\; \underset{m \in \mathcal{M}}{\text{argmax}}\; F_\tau(m) \,, \tag{4.2}$$

where m_* denotes the optimally performing assignment function with respect to τ.

We note that the problem statement above is specifically written for a single-task scenario, for which only one instance of τ requires optimal configuration. In this thesis, we further consider a new setting for neural architecture search, which focuses on concurrently configuring a set of n related tasks $\boldsymbol{\tau} \triangleq \{\tau_1, \tau_2 \ldots \tau_n\}$. We state the problem as follows:

Definition 9 (Multi-task Neural Architecture Search) *Let $\boldsymbol{\tau} \triangleq \{\tau_1, \tau_2 \ldots \tau_n\}$ be a set of neural architecture search tasks such that every task $\tau_i \triangleq \{\mathcal{G}, \mathcal{O}, \mathcal{D}_i\}$ shares the same search space $\mathcal{G}$ and the operator set $\mathcal{O}$, but differs from one another by the given task data $\mathcal{D}_i$. Using the same notation of $\mathcal{M}$ in Definition 8 and let $\mathbf{m} = \{m_1, m_2 \ldots m_n\}$ be a collection of mappers in $\mathcal{M}$, where each m_i corresponds to a candidate architecture for task τ_i. The performance measuring function for the multi-task NAS problem is then given by averaging the per-task performance:*

$$F_{\boldsymbol{\tau}}(\mathbf{m}) \;\; \triangleq \;\; \frac{1}{n}\sum_{i=1}^{n} F_{\tau_i}(m_i) \, . \tag{4.3}$$

Finally, we state the multi-task NAS objective as the following optimization task:

$$\mathbf{m}_* \;\; = \;\; \underset{\mathbf{m} \subset \mathcal{M}}{\operatorname{argmax}} \, F_{\boldsymbol{\tau}}(\mathbf{m}) \, . \tag{4.4}$$

4.2 Related work

The NAS literature can generally be classified into two separate lines of research: (1) search space design and (2) search strategy. As briefly mentioned above, this thesis will adopt the over-parameterized search space proposed by Bender et al. [6] and focus on developing new search strategies under this paradigm. This section provides a brief summary of various NAS methods in this direction (which we will simply refer to as NAS from this point onward) and some preliminary works in the multi-task NAS domain.

4.2.1 Cell-based over-parameterized search space

Most NAS methods converge on a cell-based organization of the over-parameterized network search space that was inspired by Pham et al. [68]. This entails factoring the search space graph $\mathcal{G}$ into a linear chain of modular cell blocks, each serves as an over-parameterized network by itself. Every cell has its own source and sink node, which respectively receives the output of its previous cell and forwards its own output to the next cell. The NAS outcome is then the combined edge selection for all cells in $\mathcal{G}$, which is then achieved by specific search strategies.

4.2.2 Scheduled dropout

One major challenge of NAS is the expensive cost of evaluating candidate architectures, which involves sufficient training of the selected layer weights. Bender et al. [6] proposes to isolate this bottleneck from the performance measuring function through optimizing the entire over-parameterized network $\mathcal{G}$ once before conducting edge selection, thus avoiding the repeated training cost. The cost and the stability of optimizing such a giant model are kept manageable via a steadily increasing dropout rate (i.e., the probability of zeroing out the output of a neuron) as the training progresses. Bender et al. [6] conducts random search to find the best pruning of the trained $\mathcal{G}$, but this step can easily be extended with other black-box optimization methods.

4.2.3 Continuous relaxation

A main cause for inefficiency in NAS is often due to the intractability of the edge selection problem, which can be written as a high-dimensional integer programming objective. Explicitly, we cast the output domain of the mapping function m as the space of one-hot vectors with dimension $|\mathcal{O}|$ and express the computation at edge $e = (v, v')$ as:

$$z_{v'} \;=\; \sum_{i=1}^{|\mathcal{O}|} m_i(e) \cdot o_i(z_v) \;, \tag{4.5}$$

where o_i denotes the i^{th} operator in $\mathcal{O}$, $m_i(e) \in \{0, 1\}$ denotes the i^{th} entry of the one-hot vector $m(e)$, i.e., $\sum_{i=1}^{|\mathcal{O}|} m_i(e) = 1$. Liu et al. [55] then proposes the to alleviate this problem via approximating $m(e)$ with a continuous vector $\bar{m}(e) \in \mathbb{R}^{|\mathcal{O}|}$, called the *operator mixing weights* of e. We can then rewrite the above computation using a softmax operator:

$$z_{v'} \;\simeq\; \sum_{i=1}^{|\mathcal{O}|} \frac{\exp(\bar{m}_i(e))}{\sum_{j=1}^{|\mathcal{O}|} \exp(\bar{m}_j(e))} \cdot o_i(z_v) \;. \tag{4.6}$$

In this manner, the computation induced by any candidate architecture can be expressed entirely with continuous transformations that are fully differentiable with respect to all $\{\bar{m}(e)\}_{e \in \mathcal{E}}$ and the corresponding layer weights, hence reducing the NAS problem to a gradient-based optimization task. Xie et al. [81] subsequently proposes a similar approach that instead replaces the softmax operator in Eq. (4.6) with a Gumbel-softmax operator [42], thus obtaining candidate architectures with mixing weights that converge to samples of a categorical distribution (i.e., when its temperature parameter is annealed to 0), which better approximates Eq. (4.5). Hu et al. [35] further exploits this set up to perform differentiable edge sampling using the straight-through trick [42], thus avoids the expensive cost of training all layer weights at once.

Federated NAS

More recently, the NAS problem is also considered in the federated learning context [61]. This line of research studies a variant of the multi-task NAS problem introduced above, where the task data $\mathcal{D}_i$ are assumed to be privately hosted and cannot leave their respective silos. To address this problem, He et al. [26] adopts the differentiable framework by Liu et al. [55] to perform local NAS for each task τ_i and periodically synchronizes these local architectures via averaging both their operator mixing weights and layer weights. Nonetheless, we note that this aggregation scheme will mandate all local nodes to converge at a single architecture, which is not necessarily optimal when their respective tasks are heterogeneous. We will therefore focus on solving exactly the multi-task NAS objective given in Definition 9, which instead seeks to efficiently optimize an entire collection of architectures to optimally address each task.

4.3 Motivation

This section now focuses on a new framework to address the *multi-task* NAS problem introduced above, particularly in the federated learning (FL) setting [61] where task data are not allowed to leave its computing node. While this seems like a straight-forward extension of the *single-task* NAS problem in Definition 8, naïvely applying *single-task* NAS frameworks multiple time on each target task is often a prohibitively expensive strategy. In addition, independently executing these *single-task* NAS instances also implies poor utilization of available information, especially when the target tasks are correlated. Although combining local information will trivially enable knowledge sharing among tasks, this strategy is not applicable to the FL setting due to its inability to preserve data privacy.

To address this problem scenario, the next immediate approach is to apply existing FL strategies on the *single-task* NAS objective to construct a privacy-preserving optimization scheme that combines knowledge across computing nodes. Generally, the FL-NAS objective can be stated as:

$$F_{\mathcal{T}}(m) \triangleq \frac{1}{n}\sum_{i=1}^{n} F_{\mathcal{T}_i}(m) \,, \tag{4.7}$$

where m denotes a candidate architecture, represented as in Definition 8 and $F_{\mathcal{T}_i}$ denotes the performance measuring function for client i. We note that this objective is slightly different from the *multi-task* objective given in Definition 9, as it assumes a single architecture m will suffice for all participating clients. This is, however, often sub-optimal for situations where tasks are not highly correlated [32, 48, 83]. To address this limitation, a common approach is to locally fine-tune the obtained architecture using private data until the personalized architectures can sufficiently address their respective tasks. The cost of doing so, however, might vary from client to client as the FL-NAS objective does not take into account the distance between each task-specific optimal architecture and the consensus architecture on the candidate manifold. More explicitly, there might exist an architecture that does not minimize the above objective, but has significantly larger margins of improvement upon fine-tuning with task-specific data.

Orthogonal to the NAS domain, multi-task learning has previously been considered by the meta learning framework [20, 21] to rectify the above problem in a general optimization setting. This is achieved via explicitly accounting for subsequent fine-tuning steps in the objective. Explicitly, applying this framework, Eq. (4.7) can be adapted as:

$$F_{\mathcal{T}}(m) \triangleq \frac{1}{n}\sum_{i=1}^{n} F_{\mathcal{T}_i}\left(m - \lambda \nabla_m F_{\mathcal{T}_i}(m)\right) \,, \tag{4.8}$$

in which each local model instead anticipates the performance of the candidate architecture with one additional gradient update step $\lambda \nabla_m F_{\mathcal{T}_i}(m)$. This anticipation guides the discovery of architecture with larger improvement margins. However, optimizing Eq. (4.8) with standard gradient-based approaches will require computing the computationally expensive Hessian term $\nabla^2_m F_{\mathcal{T}_i}(m)$, where m comprises the weights of all cells in the over-parameterized network.

To address this shortcoming, we propose a new algorithm which extends Eq. (4.8) with two key ideas. First, we propose a more practical knowledge organization, which factorizes the cell-based over-parameterized network into two disjoint components called the base stack and the personalized stack respectively. The base stack contains the majority of cells in the search space and is tasked to learn a common data embedding that is useful to all tasks. On the other hand, the personalized stack contains the remaining cells and is designated as the architecture component to be personalized by each local task. This hierarchical assumption helps to significantly reduce the computational cost of the meta-learning update, since the second-order gradient computation is now confined to a small fraction of the weight vector.

To compensate for the reduced expressiveness of the personalization step, we further introduce a more fine-grained personalization scheme which will enable architectures to be recommended on a *per-input* basis. In particular, we propose a modification of the continuous relaxation scheme for over-parameterized search space NAS, in which each edge-wise operator mixing weights are explicitly conditioned on the input vector (or batch input tensor) via a parameterized mapping function. A key advantage of this context-aware design is that it implicitly addresses the personalization scenario where the target tasks are related through an overlapping mode of their data distribution. For example, consider a scenario with two handwritten digit classification tasks with many overlapping labels (e.g., classifying digits $\{1, 3, 5\}$ vs. $\{3, 5, 7\}$), in which it will be intuitively information-efficient for both tasks to share a similar architecture that can distinguish between label 3 and 5.

4.4 Technical Approach

Our method, FEDPNAS, adopts a similar over-parameterized architecture space and continuous relaxation approach as suggested by Hu et al. [35]. As motivated above, we further seek to facilitate efficient personalization via factorizing the architecture space into: (1) a base component that is shared among all client models; and (2) a personalized component, which will be adapted to specifically address each task. An overview of our architecture space is given below in Fig. 4.1 and its full specification is described in Appendix D. Each candidate architecture in this space is fully specified by the concatenated layer weights W of the over-parameterized architecture and the concatenated operator mixing weights Π (i.e., combining over all cells and edges). In our formulation, Π is in turn specified by a collection of edge-wise mapping functions that compute a mixing weight vector given the initial input. For clarity, we use the notations $m \equiv (W, \Pi) \equiv (W_b, W_p, \Pi_b, \Pi_p)$ to reflect the search space factorization, where (W_b, Π_b) and (W_p, Π_p) denote all trainable parameters in the base component and personalized component respectively. For convenience, we additionally define $\theta_b = (W_b, \Pi_b)$ and $\theta_p = (W_p, \Pi_p)$.

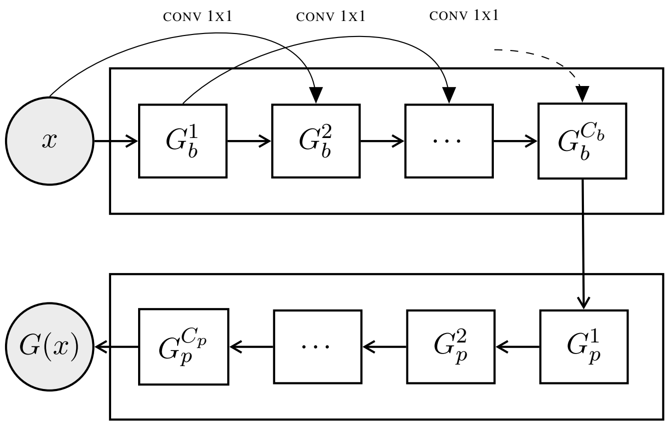


Figure 4.1: Feature mapping induced by the component stacks of our architecture space. Each cell in the base stack receives outputs from two previous cells, whereas each cell in the personalized stack receives outputs from only one previous cell.

Our algorithm is then composed of a *federated* phase and an *adaptation* phase. In the *federated* phase, we rewrite the Meta-NAS objective in Eq. (4.8) to reflect our factorized architecture space as:

$$F_\tau(\theta_b, \theta_p) \triangleq \frac{1}{n} \sum F_{\tau_i}\left(\theta_b, \theta_p - \lambda \nabla_{\theta_p} F_{\tau_i}(\theta_b, \theta_p)\right) . \tag{4.9}$$

We then derive the gradient of this personalized NAS objective in Appendix D and propose using a first-order Taylor approximation to efficiently estimate the Hessian terms that arise in the derivative. Subsequently, in the *adaptation* phase, we fix the learned parameters θ_b of the base component and proceed to fine-tune the specific component for each task using iterative gradient descent update. That is:

$$\theta_p^{t+1}(\tau_i) \triangleq \theta_p^{t+1}(\tau_i) - \lambda \nabla F_{\tau_i}\left(\theta_b^0, \theta_p^t(\tau_i)\right) , \tag{4.10}$$

where $\theta_p^t(\tau_i)$ denotes the parameters of the specific component of task τ_i at the t^{th} adaptation step and $\theta_p^0(\tau_i)$, θ_b^0 denote the parameters obtained at the end of the *federated* phase. To complete the specification of our algorithm, we further describe the context-aware mapping functions that generate operator mixing weights in Appendix D.

4.5 Results

We compare against several different architecture search and federated learning benchmarks, such as FEDDSNAS [26] and FEDAVG [61]. All of our empirical studies are conducted on two image recognition datasets: (a) the CIFAR-10 dataset [47] which aims to predict image labels from 10 classes given a train/test set of 50000/10000 colour images of dimension 32×32 pixels; and (b) the MNIST dataset [49] which aims to predict handwritten digits (i.e. 0 to 9) given a train/test set of 60000/10000 grayscale images of dimension 28×28 pixels. We compare two variants of our framework, CA-FEDPNAS (with context-aware operation sampler) and FEDPNAS (without the operation sampler), against: (a) FEDAVG of a fixed architecture to justify the need for NAS in FL; (b) FEDDSNAS - which extends DSNAS to the FL setting; and finally (c) CA-FEDDSNAS, which extends FEDDSNAS with our context-aware sampler.

To demonstrate the performance of FEDPNAS, we conduct three empirical studies. First, we design a control experiment to test our framework on heterogeneous tasks and demonstrate the necessity of architecture personalization. We simulate this scenario via applying different transformations to each client dataset. We show that on both datasets, both methods with our context sampler CA-FEDPNAS and CA-FEDDSNAS converge much faster than their counterparts without the sampler component. The performance gap is significant on the more difficult CIFAR-10 dataset. This shows that the contextual information helps to quickly locate regions of high-performing architectures, especially on similar inputs.

Second, we investigate the respective performances of CA-FEDPNAS and and FEDDSNAS on tasks with varying levels of heterogeneity. We simulate this by using varying the complexity level of data transformations, such as small-angle rotations (easy) or hue jitter/large-angle rotations (hard). We observe that our method CA-FEDPNAS achieves better performance, especially on tasks with higher heterogeneity among clients. This clearly shows the importance of architecture personalization when the training tasks are significantly different and justifies our research goal.

Finally, we assess the quality of the pre-adaptation architecture distributions respectively discovered by CA-FEDPNAS and FEDDSNAS. In particular, we use these learned distributions to generalize to completely unseen tasks (i.e., tasks that are not observed in the federated learning phase). To simulate this scenario, during the evaluation phase, however, we supply each local client with 2000 test images subjected to completely unseen transformations. CA-FEDPNAS outperformed other benchmarks without any further finetuning, and this performance gap significantly increases only with 100 iterations of retraining. This implies that CA-FEDPNAS more accurately captured the broad similarity of the task spectrum and requires minimal additional information to successfully adapt to unseen tasks.

The details of our empirical study is given in Appendix D. Our work has been published to the New Frontiers in Federated Learning: Privacy, Fairness, Robustness, Personalization and Data Ownership (NFFL) workshop in the Conference on Neural Information Processing Systems (NeurIPS) 2021.

Chapter 5

Other work and future directions

5.1 Other algorithmic selection problems for kernel-based methods

Orthogonal to the approach presented in this thesis, we have also investigated another algorithmic design problem for GP. The classical GP algorithm [70] trains and runs in $\mathcal{O}(n^3)$ complexity, where n is the size of the training data. Due to this prohibitively expensive cost, many sparse approximations have been employed to improve the scalability of GP in practical use cases. These methods, which are broadly referred to as *sparse GPs* (SGP), usually focus on optimizing a compact set of inducing data points which serve as sufficient statistics for the training data [25, 28, 33]. Nonetheless, there have been little study on how large should this inducing set be for the SGP predictions to closely approximate that of GP.

We developed a set of conditions for the training data such that the sparse spectrum GP (SSGP) method [25], only need a compact inducing set (i.e., the set of spectral frequencies) to closely approximate GP prediction. This is the first theoretical study regarding the approximation quality of SSGP. Based on this theoretical understanding, we further develop a practical approach to *recondition* the training data using a variational autoencoder (VAE) approach [45]. We showed that our VAE-transformed data exhibit the proposed conditions and achieve better sample complexity than vanilla SSGP. Our analysis and method has been published at the Conference on Neural Information Processing Systems (NeurIPS) 2020 [29].

In addition, previous chapters have considered both architecture design for Artificial neural networks (ANN) and kernel design for Gaussian processes (GP). Recent work has shown that ANNs are equivalent to GP in the infinite-width limit, and that the evolution of an ANN during training can also be described by a neural tangent kernel (NTK) [39]. This provides a theoretical connection between two major domains of AAD and reveals a future direction on an unification approach for algorithmic design.

5.2 Designing sequence sketches beyond the minimizer method

In Chapter 3, we have considered the sketch design problem grounded in the minimizer [59, 73] approach. This has been extended to unify the syncmer approach [16, 74] under the masked minimizers method, which is currently being peer reviewed for the conference on Research in Computational Molecular Biology (RECOMB) 2023 [30]. In the remaining timeline of this dissertation, we will continue to investigate other algorithmic design problems in the sequence sketching domain. One example of such problems including configuring the local scheme [60], which employs a local k-mer ordering per (w, k)-window instead of a global ordering as in the minimizer approach. Alternatively, we also consider going beyond the k-mer sampling constraint, i.e., can we sketch the sequence using variable-sized substrings or continuous embedding that allows recovery of the original sequence?

5.3 Evolution-based meta learning for heterogeneous multi-task AAD

In Chapter 4, we have considered the multi-task AAD scenario grounded in the Federated NAS problem. We showed that the meta-learning paradigm can be naturally extended to solve AAD optimization tasks with several practical improvements. However, this approach relies on the assumption that all tasks sampled from the given task distribution must be sufficiently similar, such that a single base model can be quickly adapted to solve each of them. In practice, such an assumption is often inappropriate given a long-tailed or multi-modal task distribution. In fact, high task diversity/heterogeneity has been shown to limit the performance of meta learning. For example, Chen and Zhang [10], Iwata and Kumagai [38] have recently shown that standard meta learning is not effective on tasks with different feature spaces.

To address this problem, existing approaches [10, 38] employ various task embedding strategies to implicitly cluster sampled tasks into different localities of the task manifold. These task embedding modules, however, are solely learned from the sampled task data and are assumed to stay static throughout the meta learning step, hence do not account for subsequent updates of the meta model hypothesis. We instead argue that an accurate task clustering scheme must also account for the underlying function that generates the data, and therefore needs to be concurrently updated together with the learning model.

In the remaining timeline of this thesis, we will investigate a new approach to address this learning scenario. Our preliminary idea is to adopt an evolving ensemble of meta-models to accurately reflect the multi-modality of the task landscape. Each meta model in the ensemble represents a prototype that implicitly defines a cluster of tasks. Cluster membership is then assigned based on a fitness metric that measures task similarity without learning an embedding. In particular, our fitness metric $F(\mathcal{T}, \mathcal{P})$ between a new task $\mathcal{T}$ and a prototype $\mathcal{P}$ will directly measure the agreement between the objective gradients of $\mathcal{T}$ and that of the previous tasks used to update $\mathcal{P}$. Intuitively, we hypothesize that $\mathcal{P}$ is fit to solve $\mathcal{T}$ when all gradients point in a similar direction, and is unfit otherwise. This fitness measures will determine which meta model will get to update with respect to an incoming batch of tasks. We also plan to devise a new evolutionary mechanism to replace obsolete meta models in the ensemble. Explicitly, analogous to a cross-over operator in evolutionary optimization, we will let meta models that are not frequently matched to new tasks distill knowledge from more successful models in the ensemble.

5.4 Thesis timeline

Timeline	**Milestones**
November 2022	Proposal Submission
November 2022 - January 2023	Revise Chapter 3 based on conference review
December - January 2022	Proposal Presentation
November 2022 - February 2023	Conducting research direction proposed in Section 5.3
February 2023	Submit research work to peer-review conference
February 2023 - May 2023	Thesis writing
Apr 2023 - May 2023	Revise chapter 4 based on conference review
June 2023 - July 2023	Thesis submission/defense

Table 5.1: Proposed timeline of thesis

Appendix A

Kernel Selection Appendix

A.1 Bayesian Generative Search

Similar to Lu et al. [57], we reformulate the discrete optimization problem as a BO task over a latent representation space, which allows us to bypass both the need to select an initial set of active candidates [15] and to rely on heuristic methods for exploring new candidates [58]. However, instead of learning a direct embedding, we implicitly encode composite kernels as output of a parameterized *open-ended recurrent generator*. While Lu et al. [57] focuses on learning a mapping between latent representations and kernel expressions, our method, DTERGENS, learns a mapping between the latent coordinates and the *infinitely long* kernel expression generative trajectories. This main difference helps to avoid placing an explicit upper limit on the expression length, thus ensuring sufficient expressiveness of the candidate set.

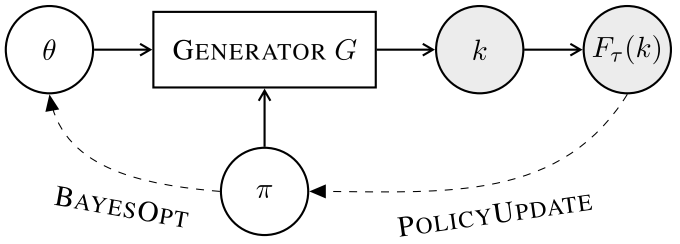


Figure A.1: The generic workflow of DTERGENS. Given policy π, we employ BO to obtain generative weight candidate θ (Section A.1.3). Using the observed generative trajectory, we alternately update the policy distribution (Section A.1.4).

The VAE decoder in [57], which sequentially generates the next operand and operator given the current expression, is pre-trained with the VAE encoder using randomly sampled kernel expressions. In contrast, our generator explicitly captures the *sum-of-product* structure through a nested recursion procedure, thus is capable of generalizing the composition rules to create new expressions. To prevent the generation of infinitely long and computationally expensive expressions, we learn a stochastic early stopping policy, which determines the best performing stopping point on any generative trajectory. This method enables the exploration of arbitrarily complex expressions and is the first selection method that places no further structural restriction on the search space.

Both the generative parameters and the termination policy can be jointly optimized by exploiting their dynamic in the generative component. Fixing a termination policy, we devise a dynamic BO algorithm for optimizing generative parameters that is capable of adapting to the constant policy updates. Alternately, given each sample trajectory collected by the BO step, we devise an update algorithm for the policy distribution via modelling the dynamic between these two components. Together, these steps compose our main contribution, which is the DTERGENS algorithm for composite kernel selection.

We demonstrate that DTERGENS is able to produce sophisticated kernel expressions, which significantly improve the predictive performance over state-of-the-art methods on multiple benchmarks. Our results show a wider range of structures being explored by DTERGENS and faster convergence compared to other methods. Finally, we show that DTERGENS is also able to recover known well-performing kernels on artificially designed predictive tasks.

A.1.1 Reformulating Kernel Selection

Directly optimizing over the discrete domain of kernel expressions as suggested by Definition 3 is challenging. To work around this, we model k as the output of a generative process $G(\theta, \pi)$ conditioned on θ and π, which controls the termination of G. The designs of G and π are given in Section A.1.2 and Section A.1.4 respectively. We approximate the original kernel selection objective in Definition 3 as:

$$\underset{k \in \mathcal{K}_{\mathcal{C}}}{\arg\max}\, F_{\Omega}(k) \quad \simeq \quad \underset{\theta \in \Theta, \pi \in \Pi}{\arg\max}\, R_{\tau}(\theta, \pi)\ , \qquad \text{(A.1)}$$

where R_τ is defined as the composite function $F_\tau \circ G$.

Our framework alternately optimizes each component π and θ while fixing the other as constant. Explicitly, given a policy π, the generative parameter θ is optimized using an adapted BO algorithm that is formulated with the conditional policy distribution $p(\pi \mid \theta)$ (Section A.1.3). On the other hand, given the BO-sampled observations, the policy distribution $p(\pi \mid \theta)$ can be updated via MLE (Section A.1.4). The outline of this workflow is illustrated in Fig. 2.2. To lay the groundwork for our algorithmic development, we will first discuss the design of our kernel generator $G(\theta, \pi)$ in Section A.1.2.

A.1.2 Open-ended Kernel Generator

A composite kernel expression, written as a sum-of-products over base kernel units in Eq. (2.4), naturally manifests as a tree with (1) a primary linear chain; and (2) several secondary linear branches that are connected to the primary chain. The secondary branches denote different products of base kernel units, whereas the primary chain denotes the summation over them. To generate such structures, we construct G by composing two nested recurrent units, which are described below. Fig. A.2 further visualizes the blueprint of this generator architecture.

Generator Overview

Our generator architecture comprises of a primary unit and a secondary unit, which respectively generate the primary chain and the secondary branches. Each unit composes of a recurrent neural network (RNN) and a termination policy, which predicts a stopping signal given the hidden state of the neural network. Every recurring step of the primary unit initiates a new secondary branch and computes an initial hidden state for the secondary RNN. On the other hand, every recurring step of the secondary unit generates a new base kernel unit on the current branch. Each unit will recur until its respective policy outputs a stopping signal given the respective current hidden state. When the primary unit terminates, we output a tree structure corresponding to a composite kernel. We give the mechanism of each component below.

Primary Unit

The primary unit $\mathcal{U}_p : \mathbb{R}^{d_p} \to \mathcal{K}_{\mathcal{C}}$ is given by the RNN G_p and the policy $\pi_p : \mathcal{K}_{\mathcal{C}} \to \mathbb{R}$. G_p has hidden dimension d_p and emits an initial hidden state for G_s. Given an arbitrary initial hidden state $h_0 \in \mathbb{R}^{h_p}$,

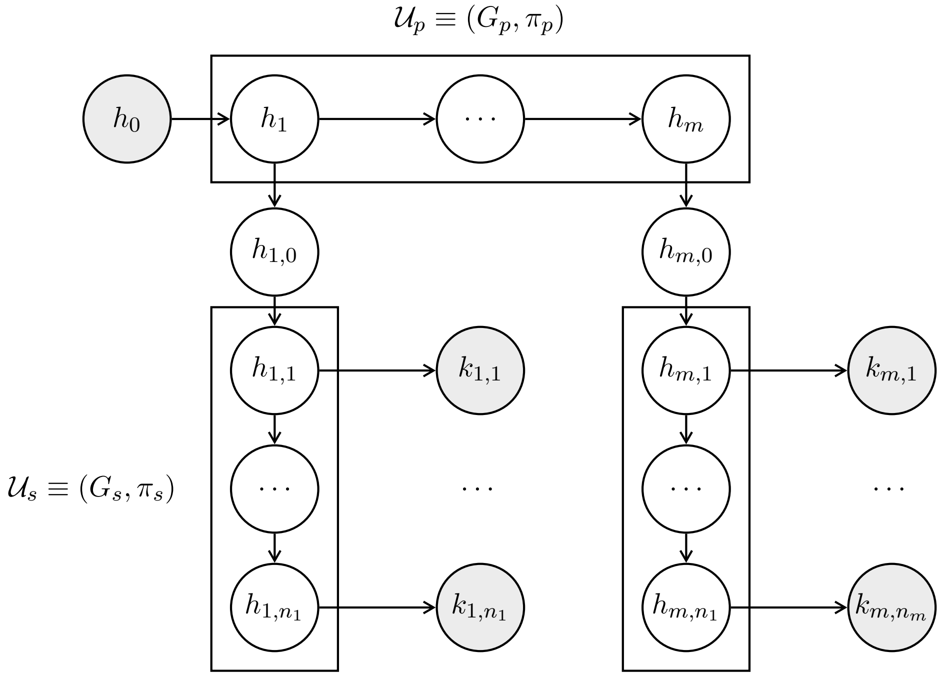


Figure A.2: Schematic of the kernel generator with nested units $\mathcal{U}_p$ and $\mathcal{U}_s$. Each component recursively computes its next hidden state and emission output using respective recurrent neural network G_s and G_p. The termination probability at each generative step is determined by policies π_p and π_s. The final candidate kernel expression is composed using Eq. (2.4).

G_p performs the following at any generative step $t \geq 0$:

- If $t = 0$, initialize candidate kernel expression $\bar{k} = 0$ (i.e., a constant function).
- Generate the next hidden state and the current emission output: $(h_{t+1}, h_{t,0}) \leftarrow G_p(h_t; \theta_t)$.
- Generate and append the t^{th} secondary branch to the current expression: $\bar{k} \leftarrow \bar{k} + \mathcal{U}_s(h_{t,0}; \theta_s, \pi_s)$.
- Query termination probability with the current intermediate expression: $\alpha \leftarrow \pi_p(\bar{k})$.
- With probability α, end the generative procedure and return $\bar{k}$ as a terminal expression.
- With probability $1 - \alpha$, set $t \leftarrow t + 1$ and repeat the procedure.

Secondary Unit

The secondary unit $\mathcal{U}_s : \mathbb{R}^{d_s} \to \mathcal{K}_{\mathcal{C}}$ is given by the RNN G_s and the policy $\pi_s : \mathcal{K}_{\mathcal{C}} \to \mathbb{R}$. G_s has hidden dimension d_s and emits one-hot representations of the base kernel units in $\mathcal{K}_{\mathcal{B}}$. Given an initial hidden state $h \in \mathbb{R}^{d_s}$ output by G_p, $\mathcal{U}_s$ generates a corresponding product of base kernel units. Given an initial hidden state $h_{t,0}$ produced by G_p at time t, G_s then performs the following at any inner loop generative step $t' \geq 0$:

- If $t' = 0$, initialize the inner loop kernel expression $\bar{k}_t = 1$ (i.e., a constant function).
- Generate the next hidden state and emit the current base kernel: $(h_{t,t'+1}, k_{t,t'}) \leftarrow G_s(h_{t,t'}; \theta_s)$.
- Extend current expression via multiplication: $\bar{k}_t = \bar{k}_t \times k_{t,t'}$.
- Query the termination probability $\beta \leftarrow \pi_s(\bar{k} + \bar{k}_t)$.
- With probability β, end the generative procedure for the current branch and return $\bar{k}_t$.
- With probability $1 - \beta$, set $t' \leftarrow t' + 1$ and repeat the procedure.

Termination Policy

Let $\tau = \{\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_n\}$ denote the set of training inputs specified by the learning instance Ω, as introduced in Definition 3. The termination policies π_p and π_s, respectively parameterized by the neural networks γ_p and γ_s, are given as follows:

$$\begin{aligned} \pi_p(k; \gamma_p, \tau) \quad &\triangleq \quad \sigma\left(\sum_{i,j \in [n]} \gamma_p(\mathbf{x}_i) \cdot \gamma_p(\mathbf{x}_j) \cdot k(\mathbf{x}_i, \mathbf{x}_j)\right) \ , \\ \text{and} \quad \pi_s(k; \gamma_s, \tau) \quad &\triangleq \quad \sigma\left(\sum_{i,j \in [n]} \gamma_s(\mathbf{x}_i) \cdot \gamma_s(\mathbf{x}_j) \cdot k(\mathbf{x}_i, \mathbf{x}_j)\right) \ , \end{aligned} \qquad (A.2)$$

where $\sigma(t) \triangleq 1/(1 + \exp(-t))$ denotes the sigmoid activation function. This data-driven parameterization serves to model the task-specific termination rules conditioned on observation task data, which implies that different expression lengths are required for different tasks. We further model the interaction between generative weights θ and termination policies π using the conditional distributions $p(\gamma_s \mid \theta)$ and $p(\gamma_p \mid \theta)$, which in turn are used to construct a kernel function (that models the covariance between composite kernels) of an adapted BO routine (Section A.1.3). This allows us to sample policies given a candidate generative weight and distills finite kernel expression from the potentially infinite trajectory. On the other hand, the adapted BO routine will collect the full trajectory of kernel generation per iteration (consisting of all intermediate expressions until the termination point). This is then used to optimize the above conditional distribution via minimizing a heuristic loss function, thus allowing us to alternately learn the dynamics between θ and π (Section A.1.4).

A.1.3 Generative Parameter Optimization

In this section, we detail our BO algorithm to optimize the generative weight θ. Formally, fixing a policy distribution $\bar{\pi}$ whose parameterization $\bar{\gamma}$ follows the conditional distribution $\bar{p}(\gamma \mid \theta) \triangleq \bar{p}(\gamma_p \mid \theta)\bar{p}(\gamma_s \mid \theta)$, the partial objective can be rewritten as:

$$\theta^* \;=\; \underset{\theta \in \Theta}{\arg\max}\, g_{\bar{\pi}}(\theta) \;\;\equiv\;\; \underset{\theta \in \Theta}{\arg\max}\, \mathbb{E}_{\gamma \sim \bar{p}}\left[R_\tau(\theta, \pi(\gamma))\right] \,. \tag{A.3}$$

We then adopt the standard practice of BO [75] and impose a Gaussian Process (GP) [70] prior on the black-box function $g_{\bar{\pi}}$, i.e., $g_{\bar{\pi}} \sim \mathcal{GP}(\mu, k_{\mathrm{BO}})$ where μ and k_{BO} respectively denote its mean and covariance functions. The BO algorithm iteratively obtains the next best candidate θ by maximizing a surrogate acquisition function constructed from the posterior distribution of this GP. The black-box evaluation $g_\pi(\theta)$ is then used to update the GP posterior.

Note that in standard BO setting, the functional landscape is static, whereas in our formulation $\bar{\pi}$ is alternately optimized after every BO iteration, which results in a changing function $g_{\bar{\pi}}$. To account for this dynamic update of $\bar{\pi}$, we will therefore model the GP covariance using two components: (1) an intrinsic kernel component that characterizes the feature distance between the generative weights θ_i and θ_j; and (2) an extrinsic kernel component that captures the divergence of their conditional policy distribution $p(\gamma \mid \theta_i)$ and $p(\gamma \mid \theta_j)$ given the current parameterization γ_p, γ_s. Explicitly, given candidates θ_i and θ_j, the kernel distance between these candidates is given as:

$$k_{\mathrm{BO}}(\theta_i, \theta_j) \;\;\triangleq\;\; k_{\textsc{Policy}}(\theta_i, \theta_j) \cdot k_{\textsc{Generator}}(\theta_i, \theta_j) \,, \tag{A.4}$$

where $k_{\textsc{Generator}}$ is given by the standard squared exponential kernel and $k_{\textsc{Policy}}$ is given by the symmetric KL divergence between the policy distributions conditioned on θ_i and θ_j respectively:

$$k_{\textsc{Policy}}(\theta_i, \theta_j) \;\;\triangleq\;\; \mathbb{KL}\Big(p(\gamma \mid \theta_i) \parallel p(\gamma \mid \theta_j)\Big) \,. \tag{A.5}$$

A.1.4 Optimizing Policy Distribution

We use the same notation γ to refer to both policy γ_p and γ_s in this section due to their symmetry. We first give our parameterization for the policy distribution $p(\gamma \mid \theta)$ using a standard Bayesian neural networks:

$$p(\gamma \mid \theta) \quad \sim \quad \mathcal{N}(\gamma_p; \mu(\theta), \Sigma(\theta)) \;, \tag{A.6}$$

where μ and Σ are deep neural networks that respectively generate the mean and covariance of the distribution. This section then details an update iteration of μ and σ given a new candidate weight θ (derived from maximizing the BO acquisition function) and its corresponding kernel expression $k = \sum_{t=1}^{m} \prod_{t'=1}^{n_t} k_{t,t'}$ where $k_{t,t'} \in \mathcal{K}_{\mathcal{B}}$. Since the generative trajectory encoded by θ is infinite, there is no analytical method to compute its optimal set of stopping points. However, we can approximate the optimal stopping point in this trajectory by finding the best performing intermediate kernel expression on the observed finite trajectory. Explicitly, let $\mathcal{S} = \{\{\bar{m}, \bar{n}_1, \bar{n}_2 \ldots \bar{n}_m\} \mid \bar{m} \leq m, \forall t \in [\bar{m}] : \; \bar{n}_t \leq n_t\}$ be the set of all possible intermediate sets of stopping points that precede k, we define the *hindsight estimation* of k as:

$$\begin{aligned} k^* &= \sum_{t=1}^{m^*} \prod_{t'=1}^{n_t^*} k_{t,t'}, \quad \text{where} \\ \{m^*, n_1^* \ldots n_{m^*}^*\} &\triangleq \underset{\{m', n_1', \ldots n_m'\} \in \mathcal{S}}{\arg\max} \; F_\tau \left(\sum_{t=1}^{m'} \prod_{t'=1}^{n_t'} k_{t,t'} \right) \;, \end{aligned} \tag{A.7}$$

and let K_τ^* denote the covariance matrix induced by k^* on training inputs the $\mathcal{D}$ described in τ. We argue that high-performing kernels likely produce covariance matrices that are similar to K_τ^* , which motivates the following loss function with respect to current candidate weight θ :

$$\begin{aligned} \mathcal{L}_\theta(\mu, \sigma) &= \mathbb{E}_{\gamma \sim \mathcal{N}(\mu(\theta), \sigma(\theta))} \Big[\langle G(\theta, \pi(\gamma)), k^* \rangle_\tau \Big] \\ &\simeq \frac{1}{r_\gamma} \sum_{i=1}^{r_\gamma} \langle G(\theta, \pi(\gamma_i)), k^* \rangle_\tau \end{aligned} \tag{A.8}$$

where r_γ denotes the number of γ samples drawn from the conditional distribution; $\pi(\gamma_i)$ denotes the termination policy parameterized by γ_i drawn from the conditional policy distribution; $G(\theta, \pi(\gamma_i))$ denotes the kernel expression generated by G with weight θ and policy $\pi(\gamma_i))$; and $\langle k, k' \rangle_\tau \triangleq \|K_\tau - K'_\tau\|_{\text{Fro}}$ denotes the Frobenius norm of the difference between the covariance matrices induced by kernel functions k and k'. This loss function, however, does not have an analytical gradient with respect to μ and

σ as it requires simulation to compute. To optimize for μ and σ, we first employ the random gradient estimation technique [65], which approximates gradient at a point by evaluating the expected gradient of its υ-Gaussian smoothing. In particular, we derive our randomized gradient estimation for μ as follows:

$$\begin{aligned}
\nabla_\mu \mathcal{L}_\theta(\mu,\sigma) &\simeq \nabla_\mu \mathbb{E}_{\upsilon\sim\mathcal{N}(0,\mathbf{I})}\left[\ell_\theta(\mu+\upsilon,\sigma)\right] \\
&= \mathbb{E}_{\upsilon\sim\mathcal{N}(0,\mathbf{I})}\left[\ell_\theta(\mu+\upsilon,\sigma)\upsilon\right] \\
&\simeq \frac{1}{r_\upsilon}\sum_{j=1}^{r_\upsilon}\ell_\theta(\mu+\upsilon_j,\sigma)\upsilon_j \\
&\simeq \frac{1}{r_\gamma r_\upsilon}\sum_{i=1}^{r_\gamma}\sum_{j=1}^{r_\upsilon}\left\langle G(\theta,\pi(\mu_i+\upsilon_j,\sigma_i),k^*\right\rangle_\tau \ , \qquad \text{(A.9)}
\end{aligned}$$

where r_υ denotes the number of υ samples drawn from the standard Gaussian distribution $\mathcal{N}(0,\mathbf{I})$ and we have rewritten $\pi(\gamma_i) = \pi(\mu_i,\sigma_i)$ to clearly show the perturbed component μ_i in the estimation. Similarly, the gradient estimation for σ is given as:

$$\nabla_\sigma \mathcal{L}_\theta(\mu,\sigma) \simeq \frac{1}{r_\gamma r_\upsilon}\sum_{i=1}^{r_\gamma}\sum_{j=1}^{r_\upsilon}\left\langle G(\theta,\pi(\mu_i,\sigma_i+\upsilon_j),k^*\right\rangle_\tau \ . \qquad \text{(A.10)}$$

These estimations allow us to update μ and σ via the gradient descent algorithm, which complete the specification of our policy update.

A.2 Experiments

This section evaluates and reports the empirical performance of our kernel selection framework DTERGENS on a synthetic kernel recovery task and kernel selection for regression on three real-world datasets:

- The DIABETES dataset [17] contains 442 diabetes patient records (i.e., inputs) with 10 variables: age, sex, body mass index, average blood pressure and six blood serum measurements. The target output variable is a quantitative measure of disease progression one year after baseline.
- The MAUNA LOA dataset [44] measures monthly average CO_2 concentration (in ppvm) of air samples at the Mauna Loa Observatory over 42 years (i.e., 504 observations in total).
- The PROTEIN dataset [69] features 45730 observations of protein tertiary structures, each records 9 physicochemical properties of a protein. The target output variable is the size of the protein residue in kDa.

To demonstrate the performance of DTERGENS, we compare our method with the following benchmarks: (a) random search over the space of kernels with max length $L \leq 10$ (baseline); (b) SVO: Structure Variationally-Encoded Optimization [57], for which we train the VAE component using 25000 randomly generated kernel expressions with max length $L \leq 10$ (to show the advantage of generative search); and (c) our own algorithm with no stopping policy and fixing expression length $L = 2, 4, 8$. For (c), the termination of the secondary component is chosen at random, the termination of primary component is guaranteed upon reaching length L, and REMBO [79] is used to optimize generative weights θ. This setting is meant to demonstrate the advantage of having adaptive termination policies for the generative components.

For all experiments, we demonstrate the performance of our framework on the black-box model Variational DTC Sparse Gaussian Process (vDTC) [28] with the following configurations: (1) 80/10/10 train-test-validation split (i.e., we use the validation fold to compute BO feedback and the test fold to evaluate final performance); (2) 100 randomly selected inducing inputs; (3) kernel hyper-parameters are optimized using L-BFGS over 100 iterations. These configurations implicitly define the learning scenario Ω, such that F_Ω is the root-mean-square-error (RMSE) of predictions on the test split, given a model trained and validated accordingly on the train split. We construct the set of base kernels with 4 different base kernel functions, as suggested by Duvenaud et al. [15]. These kernel functions, along with their learnable parameters, are defined as follows:

$$k_{\text{LIN}}(\mathbf{x}_i, \mathbf{x}_j; \sigma_n, \sigma_b, \mathbf{c}) \triangleq \sigma_n^2 (\mathbf{x}_i - \mathbf{c})^\top (\mathbf{x}_j - \mathbf{c}) + \sigma_b^2 \quad \text{(A.11)}$$

$$k_{\text{SE}}(\mathbf{x}_i, \mathbf{x}_j; \sigma_n, \ell_1, \ell_2 \ldots \ell_d) \triangleq \frac{1}{\sigma_n^2} \exp\left(\sum_{t=1}^{d} \frac{(\mathbf{x}_i^t - \mathbf{x}_j^t)^2}{\ell_t^2}\right) \quad \text{(A.12)}$$

$$k_{\text{PER}}(\mathbf{x}_i, \mathbf{x}_j; \sigma_n, \sigma_p, \ell_1, \ell_2 \ldots \ell_d) \triangleq \frac{1}{\sigma_n^2} \exp\left(\sum_{t=1}^{d} \frac{2\sin^2\left(\pi|\mathbf{x}_i^t - \mathbf{x}_j^t|/\sigma_p\right)}{\ell_t^2}\right) \quad \text{(A.13)}$$

$$k_{\text{RQ}}(\mathbf{x}_i, \mathbf{x}_j; \sigma_n, \sigma_w, \mathbf{c}) \triangleq \sigma_n^2 \left(1 + \sum_{t=1}^{d} \frac{(\mathbf{x}_i^t - \mathbf{x}_j^t)^2}{2\sigma_w^2 \ell_t^2}\right)^{-\sigma_w} \quad \text{(A.14)}$$

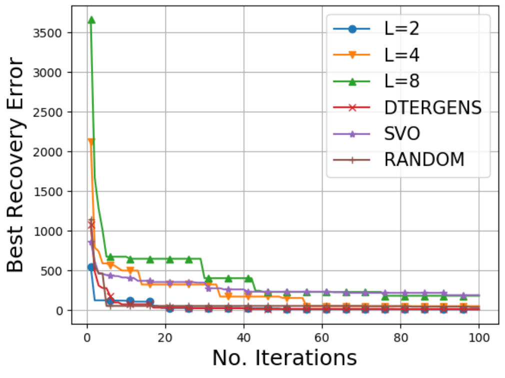
We parameterize G_p and G_s using the same RNN architecture with 4 hidden feed-forward layers. Both G_p and G_s has hidden dimension $d_p = d_s = 5$. The emission output of G_p has dimension $d_s = 5$, as G_p is tasked to generate the initial hidden state of G_s, whereas the emission output of G_s has dimension $|\mathcal{K}_\mathcal{B}| = 4$, which corresponds to the number of base kernel functions. We use ReLU activation for all nonoutput layers, softmax activation for the kernel output layer of G_s and tanh activation for the emission output layer of G_p. Finally, we optimize our RNN parameters by adapting a known high-dimensional BO method called REMBO [79] to account for the dynamic function landscape (Section A.1.3).

A.2.1 Synthetic Kernel Recovery

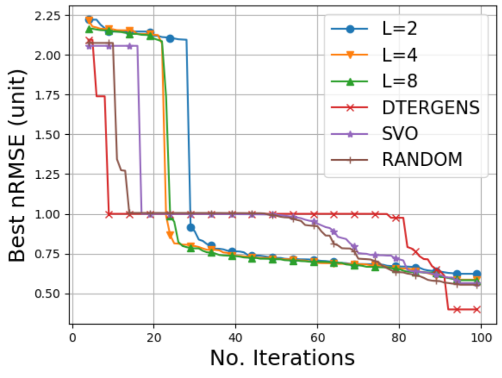
We first investigate how well various kernel selection methods recover a covariance matrix given synthetic data randomly drawn from its corresponding distribution. Unlike most real-world settings where a ground truth kernel is not known and performance evaluation relies on possibly noisy predictive accuracy, this scenario provides a ground truth for kernel selection and allows us to directly measure the success of various contending methods.

Explicitly, given an arbitrarily chosen kernel k^* (with arbitrarily initialized hyper-parameters) and n i.i.d. input observations $\tau = \{\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_n\} \subset \mathbb{R}^d$ drawn from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, we subsequently generate corresponding output observations $Y = \{y_1, y_2 \ldots y_n\}$, where $y_i \sim \mathcal{N}(0, K^*_\tau + \sigma^2\mathbf{I})$ and K^*_τ denotes the data covariance matrix induced by k^*. We then apply various kernel selection methods, including DTERGENS for vDTC prediction on this synthetic dataset and measure our recovery error for any selected kernel k by $\mathcal{L}_{\text{rec}}(k) = \|K_\tau - K^*_\tau\|_{\text{Fro}}$. Fig. A.3 (left) shows the best recovery errors achieved over a span of 100 BO iterations with 3 different ground truth kernels: (1) $k^* = k_{\text{RQ}} \times k_{\text{RQ}}$; (2) $k^* = k_{\text{PER}} \times k_{\text{RQ}} \times k_{\text{LIN}} \times k_{\text{LIN}}$; and (3) $k^* = k_{\text{LIN}} \times k_{\text{RQ}} \times k_{\text{LIN}} + k_{\text{PER}} \times k_{\text{LIN}} + k_{\text{RQ}} \times k_{\text{SE}}$.

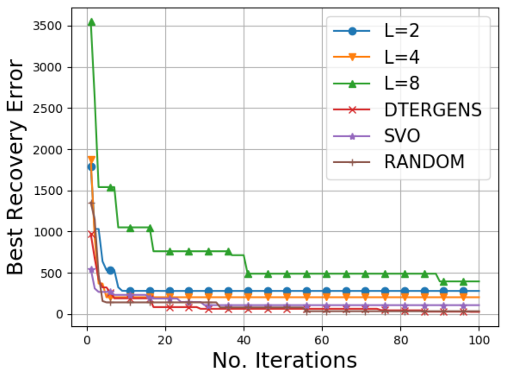
In all experiments, DTERGENS consistently achieves the lowest recovery error after 100 iterations compared to other methods. Random search performs competitively when the ground truth kernels are simple (i.e., $L = 2, 4$), hence easy to be found via randomization. On the other hadn, random search expectedly performs the worst when the ground truth kernel is longer (i.e., $L = 7$). We also observe that without the termination policy component, the performance of DTERGENS is only competitive when L is set to be roughly the length of the ground truth kernel, but otherwise outperformed by other methods. This shows the importance of adaptively learning the complexity of the kernel expression using a data driven policy. Lastly, we observe that SVO is most significantly outperformed by DTERGENS in the first experiment. We reason that this is because the number of length-2 kernels is relatively smaller in the set of training expressions for the VAE component of SVO. Thus, the trained VAE could be biased to produce



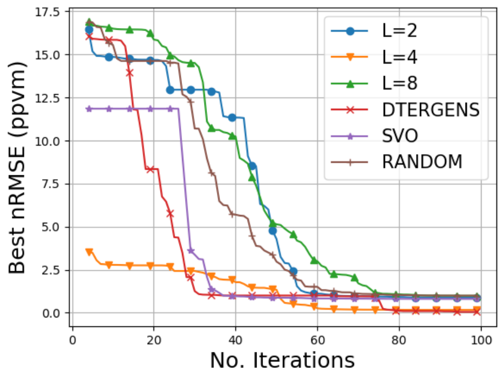
(L1) RQ $\times$ RQ



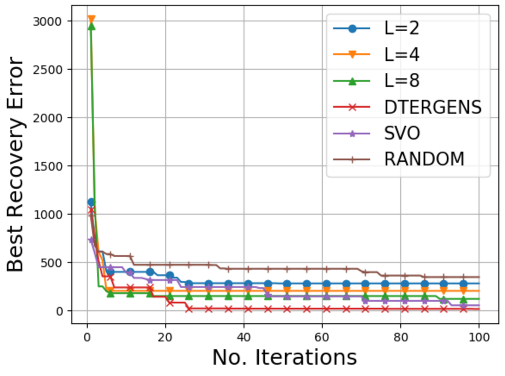
(R1) DIABETES



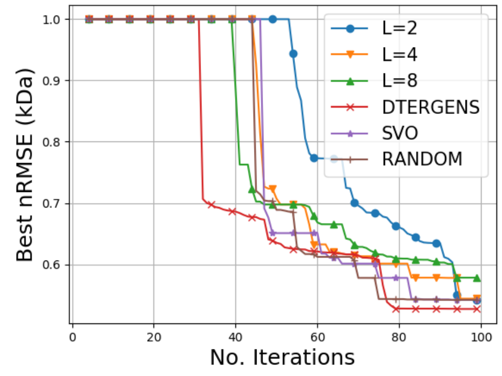
(L2) PER $\times$ RQ $\times$ LIN $\times$ LIN



(R2) MAUNA LOA

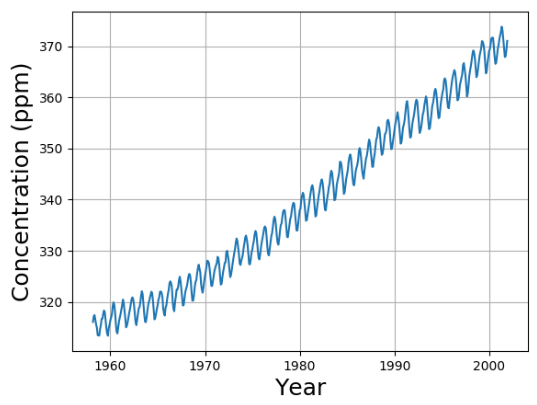


(L3) LIN $\times$ RQ $\times$ LIN $+$
PER $\times$ LIN $+$ RQ $\times$ SE

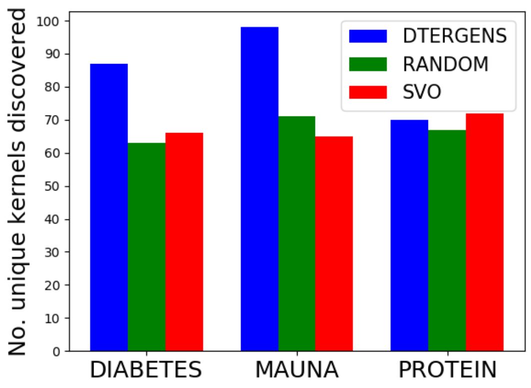


(R3) PROTEIN

Figure A.3: (Left): Best kernel recovery error over 100 iterations with various kernel selection methods on three synthetic datasets constructed from specific kernels; (Right): Best nRMSE over 100 iterations with various kernel selection methods on 3 benchmark datasets using vDTC [28].



(a)



(b)

Figure A.4: (a) The linear-periodic trend of the MAUNA dataset; and (b) number of unique kernels discovered by DTERGENS, SVO and random search on all three datasets.

longer kernels and it is more difficult for SVO to find a latent embedding that decodes to a length-2 kernel. In contrast, DTERGENS does not incur this problem because its termination policy is also learned as it collects information about the embedding space.

A.2.2 Kernel Selection for Regression Task

This section investigates the performance of kernel selection for regression tasks using vDTC [28] on DIABETES [17], MAUNA [44] and PROTEIN [69] datasets. In all experiments, we measure performance by computing the root-mean-square-error of predictions, normalized against the root-mean-square-error achieved by fixing the kernel of VDTC to be k_{SE}, which serves to demonstrate the improvement over the default choice of kernel. Explicitly, our kernel selection metric for any selected kernel k is given by:

$$\text{nRMSE}(k) \triangleq \sqrt{\frac{\sum_{i=1}^{n_{\text{test}}} \left(\bar{y}_i(k) - y_i\right)^2}{\sum_{i=1}^{n_{\text{test}}} \left(\bar{y}_i(k_{SE}) - y_i\right)^2}} \tag{A.15}$$

where $\bar{y}_i(k)$ denotes the prediction made by vDTC for test input $\mathbf{x}_i$ with selected kernel function k and y_i denotes the corresponding ground truth test output.

Fig. A.3 (right) shows the comparative performance between DTERGENS and the competing methods. Across all datasets, DTERGENS consistently obtains the best performing kernel expression. On the

PROTEIN dataset, DTERGENS also shows the fastest convergence among all competing methods. On the MAUNA dataset, DTERGENS performs competitively with $L = 4$ and both variants of DTERGENS outperform SVO. More interestingly, the best kernel found for the MAUNA dataset is $k_{\text{LIN}} \times k_{\text{PER}} \times k_{\text{PER}} + k_{\text{RQ}} \times k_{\text{PER}}$, which accurately reflects the linearly increasing periodic nature of the data (Fig. A.4a).

Fig. A.4b further compares the expressiveness of the three kernel selection methods (i.e., DTERGENS, SVO and random search), which is measured by the number of unique kernels found over 100 iterations in each method. As expected, random search consistently produces the same amount of unique expressions across all experiments. While SVO discovers approximately the same amount of unique kernels as does random search on all three datasets, it tends to outperform random search as its discovery is guided. Finally, we observe that DTERGENS consistently discovers more unique kernels and also outperforms the other methods. This finding asserts our earlier intuition on how adding expressiveness to the embedding method also helps to improve search efficiency.

Appendix B

Minimizer Sketch Design Appendix

B.1 Differentiable Learning of Minimizer Schemes

This thesis instead tackles the problem of minimizer sketch selection via directly learning a total order π using gradient-based optimization. We note that the difficulty of this task comes from two factors: (1) the search space of k-mer orderings is factorially large; and (2) the density minimizing objective is discrete. To overcome these challenges, we reformulate the original problem as parameter optimization of a deep learning system. This results in the first fully-differentiable minimizer selection framework that can be efficiently optimized using gradient-based learning techniques. The remainder of this section is organized as follows:

- Section B.1.1 defines a well-behaved search space for k-mer permutations that can efficiently leverage gradient-based optimization. This is achieved by representing k-mer orderings as continuous score assignments, output by a convolutional neural network called PRIORITYNET, whose architecture guarantees that any score assignment will correspond to a valid minimizer scheme.
- Section B.1.2 then approximates the discrete density minimizing objective by a pair of surrogate sub-tasks: (a) generating valid minimizers, which is achieved by the above PRIORITYNET; and (b) generating low density score assignments, which is achieved by another complementary neural network called TEMPLATENET. We outline the design of TEMPLATENET in Section B.1.3
- Finally, Section B.1.4 describes a surrogate loss function that measures the difference between the

outputs of these networks. Doing so results in a consensus score assignment that both corresponds to a valid minimizer and has low density on the target sequence.

B.1.1 Search Space Reparameterization

We remark that many existing methods can be seen as replacing the condition $\mathbb{I}(\kappa <_\pi \kappa')$ in Definition 4 with $\mathbb{I}(f(\kappa) <_\pi f(\kappa'))$ for arbitrary k-mers $\kappa, \kappa' \in \Sigma^k$. For example, f can be parameterized with frequency information from the target sequence [11, 40], i.e., $f(\kappa; S) \propto \sum_{j=1}^{L} \mathbb{I}(\kappa_j^k = \kappa)$; or instantiated with a UHS υ [18, 85], i.e., $f(\kappa; \upsilon) = \mathbb{I}(\kappa \notin \upsilon)$. Similar set-ups have been explored in the context of sequence-specific minimizers using a pruned UHS $\upsilon(S)$ [12] and a polar set $\zeta(S)$ [86] constructed for the target sequence. Here, we note that the notation f is overloaded to admit different parameter representations. This is mainly to highlight the unification of existing methods, and has no implication on the mathematical consistency of our formulation.

These methods can be seen as crude approximations of the total ordering π which map k-mers to a small number of discrete values and rely on a pre-determined arbitrary ordering to break ties in windows with two or more similarly scored k-mers. When collisions occur frequently, this could induce unexpected impact on the final density. Our method, DEEPMINIMIZER instead employs a continuous parameterization of f using a feed-forward neural network parameterized by weights α. f takes as input the multi-hot encoding of a k-mer and returns a real-valued score in $[0, 1]$. As continuous scores are less likely to collide, this scheme practically eliminates collisions in the resulting score assignment and thus allow recovering an exact total ordering. Explicitly, using the notion of this function f, we can subsequently rewrite the selector function in Definition 4 as:

$$m(\kappa_v^{w_k}; f) \triangleq \underset{i \in [1,w]}{\operatorname{argmin}} \, f(\kappa_i^k; \alpha) \, . \tag{B.1}$$

Further let $\mathcal{M}(\alpha)$ denote the minimizer scheme induced by applying the selector function above, the MSD problem can be written as:

$$\alpha^* = \underset{\alpha}{\operatorname{argmin}} \, D(S; \mathcal{M}(\alpha)) \, . \tag{B.2}$$

Practically, applying this network on every k-mer in S can be efficiently written as a single convolutional neural network (CNN). To differentiate this from the atomic function f, we denote the output of the

CNN as $\mathbf{f}(S;\alpha) \triangleq [f(\kappa_i^k;\alpha)]_{i\in[L]}$. We require that the score assignment induced by the CNN f to be *consistent* across different windows in order to recover a valid ordering π. Specifically, one k-mer can not be assigned different scores at different locations in S. To enforce this, we let the first convolution layer of our architecture, PRIORITYNET, have kernel size k, and all subsequent layers to have kernel size 1. This design ensures that the output entry corresponding to a k-mer is only dependent on the encoding of that k-mer alone. An illustration for $k = 2$ is given in Fig. B.1.

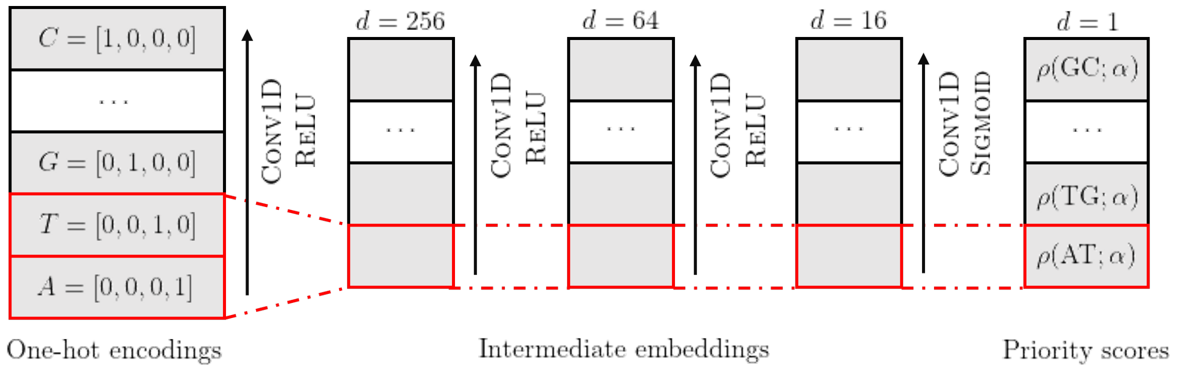


Figure B.1: Our PRIORITYNET architecture for $k = 2$, parameterized by weights α, maps sequence multi-hot encoding to priority scores through a series of 3 convolution layers with kernel size $[k, 1, 1]$ and $[256, 64, 16]$ embedding channels respectively. Fixing network weights α, the computation of any k-mer priority score is deterministic given its multi-hot encodings.

B.1.2 Proxy Objective

The density computation in Eq. B.2, however, is not differentiable with respect to the network weights. As such, α cannot be readily optimized with established gradient back-propagation techniques used in most deep learning methods. To work around this, we introduce a proxy optimization objective that approximates Eq. B.2 via coupling PRIORITYNET with another function called TEMPLATENET. Unlike the former, TEMPLATENET relaxes the *consistency* requirement and generates *template* score assignments that might not correspond to valid minimizer schemes. In exchange, such *templates* are guaranteed to yield low densities by design.

Intuitively, the goals of these networks are complementary: PRIORITYNET generates valid minimizer schemes in the form of *consistent* priority score assignments, whereas TEMPLATENET pinpoints neighborhoods of low-density score assignments situated around its output templates. This reveals an alternative optimization route where these networks negotiate towards a consensus solution that (a) satisfies the constraint enforced by PRIORITYNET; and (b) resembles a template in the output space of TEMPLATENET, thus potentially yielding low density. Let f and g denote our proposed PRIORITYNET and TEMPLATENET, respectively parameterized by weights α and β. Here g is an atomic function that maps a k-mer index to a real-valued score in $[0, 1]$, and we also denote the output of applying g on every k-mer indices as $\mathbf{g}(S; \beta)$. Last, we formalize our objective as minimizing a distance metric Δ between $\mathbf{f}$ and $\mathbf{g}$:

$$(\alpha_*, \beta_*) \quad = \quad \underset{\alpha,\beta}{\operatorname{argmin}} \, \Delta\left(\mathbf{f}(S; \alpha), \mathbf{g}(S; \beta)\right) \ . \tag{B.3}$$

We subsequently detail the full specification of our proxy objective, which includes two other components. First, Section B.1.3 discusses the parameterization of our TEMPLATENET to consistently generate templates that achieve the theoretical lower-bound density [59] on the target sequence. Section B.1.4 then discusses a practical choice of Δ to accurately capture high-performing neighborhoods of minimizers. These specifications have strong implications on the expressiveness of the solution space and directly influences the performance of our framework, as shown in Section B.2.

B.1.3 Specification of TemplateNet

The well-known theoretical lower bound $1/w$ for density implies that the optimal minimizer, if it exists, samples k-mers exactly w positions [59]. As a result, we will construct g such that $\mathbf{g}(S; \beta)$ approximates this uniform assignment pattern given any initialization of its parameter β. Proposition 1 below shows a sufficient construction of g such that $\mathbf{g}(S; \beta)$ approximately yields the optimal density.

Proposition 1 *Let $g : \mathbb{R} \rightarrow [0, 1]$ be a periodic function, with fundamental period w, such that g has a unique minimum value on every w-long interval. Formally, h satisfies:*

$$(1) : \forall t \in \mathbb{R} : g(t) = g(t + w) \qquad \textit{and} \qquad (2) : \forall i, j \in \underset{t}{\operatorname{arginf}}\, h(t),\ \exists u \in \mathbb{N} : |i - j| = uw \ .$$

Then, the template $\mathbf{g}(S; \beta) \triangleq [g(i)]_{i \in [L]}$ induces a sketch with density factor $1/w + o(1)$ on S when S is sufficiently long (i.e., $L_w \gg w^2$).

Proof: We will now express the density of S in terms of the template score assignment $g(S)$. Note that even though g may not satisfy the consistency constraint, it will still induce a k-mer sampling scheme. Let $m(\kappa_t^{w_k}) \triangleq \underset{j\in[t,t+w]}{\operatorname{argmin}}\ g(j)$ be the selector function induced by g of window $\kappa_t^{w_k}$, and let γ_t indicates the event that the t-th window picks a different k-mer than the $(t-1)$-th window. Particularly, $\gamma_1 \triangleq 1$ and $\gamma_t \triangleq \mathbb{I}(m(\kappa_t^{w_k}) \neq m(\kappa_{t-1}^{w_k}))$. Then, the density of the scheme induced by $g(S)$ is given by:

$$D(S;g) \quad = \quad \frac{1}{L_w}\sum_{t=1}^{L_w}\gamma_t\ . \tag{B.4}$$

For any value of $u \in \mathbb{N}^+$, we further define the integer interval $\mathcal{I}_u \triangleq [(u-1)w+1, uw]$. As the density of the entire sequence is simply the sum of density for each interval $\mathcal{I}_u$, it is then sufficient to derive the values of γ_t for all values of t in some arbitrary interval $\mathcal{I}_u$.

Without loss of generality, we assume $0 \in \underset{t}{\operatorname{arginf}}\ g(t)$ since this can always be achieved via adding a constant phase shift to g. As g has a period of w, this implies $\{uw \mid u \in \mathbb{N}^+\} \subseteq \underset{t}{\operatorname{arginf}}\ g(t)$, which further reduces to $\{uw \mid u \in \mathbb{N}^+\} \equiv \underset{t}{\operatorname{arginf}}\ g(t)$ when condition (2) holds. Then, it follows that $\forall t \neq uw$, we have $t \notin \underset{t}{\operatorname{arginf}}\ g(t)$. In addition, the index uw is in the window $\kappa_t^{w_k}$ by definition. Together, the above facts imply that $\forall t : m(\kappa_t^{w_k}) = uw$ and consequently $\gamma_t = 0$ for all $t \neq (u-1)w+1$, since the index uw is overlapped by $\kappa_{(u-1)w+1}^{w_k}$. For $u = 1$, we trivially have $\gamma_{(u-1)w+1} = \gamma_1 = 1$ by definition. For any $u > 1$, we have $m(\kappa_{(u-1)w}^{w_k}) = (u-1)w$ and $m(\kappa_{(u-1)w+1}^{w_k}) = uw$, which imply that $\gamma_{(u-1)w+1} = 1$. Finally, using the above derivations, we have:

$$\mathcal{D}(S;g) = \frac{1}{L_w}\left(c + \sum_{u=1}^{\lfloor \frac{L_w}{w} \rfloor}\sum_{t\in\mathcal{I}_u}\gamma_t\right) = \frac{1}{L_w}\left(c + \left\lfloor \frac{L_w}{w} \right\rfloor\right)\ , \tag{B.5}$$

where $c \triangleq \sum_{t=\lfloor \frac{L_w}{w} \rfloor w+1}^{L_w}\gamma_t$ is the remainder of the sequence that does not make up any complete interval. The second equality follows from the derived values of γ_t for $t \in \mathcal{I}_u$. Finally, using the fact that $c = L_w - \lfloor \frac{L_w}{w} \rfloor w < w$ and the sufficient length assumption $L_w \gg w^2$, we have:

$$\frac{1}{L_w}\left(c + \left\lfloor \frac{L_w}{w} \right\rfloor\right) \quad < \quad \frac{1}{w} + \frac{w}{L_w} \quad = \quad \frac{1}{w} + o(1)\ , \tag{B.6}$$

which concludes our proof. □

Note that the resulting sketch induced by g does not necessarily correspond to a valid minimizer. While this sketch has guaranteed low density, it does not preserve the sequence identity like a minimizer sketch, hence is not useful for downstream applications. However, it is sufficient as a guiding template to help PRIORITYNET navigating the space of orderings. By Proposition 1, TEMPLATENET can be as simple as $g(t) = \sin(2\pi t/w)$ to induce a near-optimal score assignment. This naïve specification, however, encodes exactly a single set of template minima (i.e., one that picks k-mers from the set of interval positions $\{w, 2w, \dots\}$), which might not be in proximity of any valid minimizer scheme. For example, consider a sequence S in which some particular k-mer uniquely occurs at positions $t \in \left\{\frac{1}{2}w, \frac{3}{2}w, \dots\right\}$. The ideal assignment would be such that minima will occur at these locations, which is impossible.

It is therefore necessary that the specification of TEMPLATENET is sufficiently expressive for Eq. B.3 to find an optimal solution. To model this family of template functions, we subsequently propose several parameterization strategies using (1) an ensemble of sinusoidal functions with integer phase shifts or (2) a Fourier series model that encodes any arbitrary sinusoidal function. We further propose an independent positional phase-delay component that can be combined with (1) and (2) to encode template functions with approximately constant period.

Ensemble Template Model

We first give a construction of a periodic model such that every k-mer position appears in at least one template encoded by its parameter space. To achieve this, we employ a linear combination of multiple sine functions with fixed integer phase shifts $\phi \in [w-1]$, each of which encodes a set of minima with a unique positional offset such as $\mathcal{T}_1 = \{0, w, 2w, \dots\}, \mathcal{T}_2 = \{1, w+1, 2w+1, \dots\}, \dots\ \mathcal{T}_{w-1} = \{w-1, 2w-1, 3w-1, \dots\}$. In particular, we define:

$$g(t;\beta) \triangleq \sigma\left(\sum_{\phi=0}^{w-1} \beta_\phi \sin\left(\frac{2\pi}{w}(t+\phi)\right)\right) , \tag{B.7}$$

where the sigmoid activation function σ ensures that $h(t)$ appropriately maps to $[0,1]$ and outputs scores on the same scale as PRIORITYNET; $\beta = \{\beta_\phi\}_{\phi=0}^{w-1}$ are optimizable amplitude parameters such that $\beta_\phi \geq 0$ and $\sum_{\phi=1}^{w} \beta_\phi = 1$. Optimizing β then determines the dominant phase shift $\phi_{\max} = \operatorname{argmax}_\phi \beta_\phi$, which in turn controls the final offset of the template minima. Additionally, allowing the amplitudes of the ensemble components to be optimizable also helps to generate sufficient slack room for matching the template scores against the priority scores.

Truncated Fourier Series Template Model

The periodic function $g(t)$ with period w can be generalized using a Fourier series, which is a linear combination of an infinite number of sine and cosine functions, whose frequencies are integer multiples of $1/w$:

$$g(t;\beta) \quad = \quad \sigma\left(\beta_0 + \sum_{r=1}^{\infty}\left[\beta_{r,1}\sin\left(\frac{2r\pi}{w}t\right) + \beta_{r,2}\cos\left(\frac{2r\pi}{w}t\right)\right]\right) \ , \tag{B.8}$$

where $\beta = \{\beta_{r,1}, \beta_{r,2}\}_{r=0}^{\infty} \cup \{\beta_0\}$ are optimizable amplitude parameters. For computational efficiency, we approximate g by a finite truncation up to the first R summands of the above Fourier series:

$$g(t;\beta) \quad \simeq \quad \sigma\left(\beta_0 + \sum_{r=1}^{R}\left[\beta_{r,1}\sin\left(\frac{2r\pi}{w}t\right) + \beta_{r,2}\cos\left(\frac{2r\pi}{w}t\right)\right]\right) \ . \tag{B.9}$$

Similar to the ensemble template model, optimizing the amplitude parameters β of this model also determines the offset of the minima locations and adds slack room to help matching against the priority score assignment. The key difference between these two template models is that the ensemble model requires all w phase shifts (and hence, all w component functions) to encode every k-mer location, whereas the Fourier model can achieve the same with a fixed value of R and remains compact even for large w. The Fourier model, however, will admit periodic functions whose minima do not coincide with integer indices, therefore condition (2) above will be less likely to hold in practice.

Positional Phase-Shift Model

By Proposition 1, all template score assignments encoded by the above β-parameterized families of functions correspond to near-optimal minimizer schemes with approximately perfect density factors. However, we note that this set of template solutions is usually unrealistic and cannot be mirrored exactly by PRIORITYNET, especially on complex problem instances with more difficult scoring constraints. For example, while the theoretical lower bound for densityis $1/w$, the actual optimal density factor attainable given a specific sequence is often considerably larger and occurs when consecutive minimizer locations are not always exactly w locations apart.

Motivated by this observation, we further extend our template model with a learnable component that adaptively adjusts the local frequencies of every encoded periodic function through adding positional noise to their phase shift parameters. That is, let $\xi(S;\gamma) \in [-1,1]^L$ be a noise generating function parameterized by γ and let $\xi_i(S;\gamma)$ be the noise value corresponding to the i^{th} k-mer. We define the (ϵ,γ)-augmented TEMPLATENET as:

$$\mathbf{g}(S;\beta,\gamma) \triangleq \left[g\left(i + \epsilon \cdot \xi_i(S;\gamma);\beta\right)\right]_{i\in[l]} , \tag{B.10}$$

where $\xi_i(S;\gamma)$ denotes the i-th entry of the noise vector. This will allow every entry in the template score assignment to be adjusted by a phase shift of up to ϵ in magnitude. When $\epsilon = 0$, this space of template functions coincides with that of the exact periodic template model, thus encodes all theoretical optimal assignments. On the other hand, as ϵ increases, more template assignments are admitted, but the optimal density guarantee becomes less certain.

B.1.4 Specification of Distance Metric

As standard practice, we first consider as our objective the ℓ^2 distance, which is given by:

$$\Delta_{\ell^2}(\mathbf{f}(S;\alpha),\mathbf{g}(S;\beta)) \triangleq \sum_{i=1}^{l}(\mathbf{f}_i(S;\alpha) - \mathbf{g}_i(S;\beta))^2 , \tag{B.11}$$

where $\mathbf{f}_i$ and $\mathbf{g}_i$ are respectively the shorthands for the i^{th} entries of $\mathbf{f}(S;\alpha)$ and $\mathbf{g}(S;\beta)$. For ease of notation, we assume that the notation β also incorporate γ in case the positional noise model is used. This metric, however, places an excessively strict matching objective at all locations along $\mathbf{f}$ and $\mathbf{g}$, which is often unnecessary. Indeed, it is sufficient to ensure that the k-mers at the selected locations are assigned lowest scores. Enforcing a perfect matching will take away the degrees of freedom needed for the proxy objective to satisfy the constraints implied by PRIORITYNET (i.e., a k-mer has to be assigned the same score at all of its occurrences).

As such, we are interested in constructing an alternative distance metric that: (a) prioritizes matching $\mathbf{f}$ and $\mathbf{g}$ around the neighborhoods of minima; and (b) allows flexible assignment at other positions to admit more solutions that meet the consistency requirement. To accomplish these design goals, we propose the following asymmetrical distance metric:

$$\Delta_{\mathcal{DM}}(\mathbf{f}(S;\alpha),\mathbf{g}(S;\beta)) \triangleq \sum_{i=1}^{L}\left[(1-\mathbf{g}_i)\cdot(\mathbf{f}_i-\mathbf{g}_i)^2 + \lambda\cdot(1-\mathbf{f}_i)^2\right] . \tag{B.12}$$

Specifically, the intuition behind the first component $(1-\mathbf{g}_i)\cdot(\mathbf{f}_i-\mathbf{g}_i)^2$ in the summation is to weight each position-wise matching term $(\mathbf{f}_i-\mathbf{g}_i)^2$ by its corresponding template score. The weight term $1-\mathbf{g}_i$ implies stronger matching preference around the minima of $\mathbf{g}$ where the template scores $\mathbf{g}_i$ are low; and vice-versa weaker matching preference at other locations where $\mathbf{g}_i$ are high. The second component $\lambda \cdot (1-\mathbf{f}_i)^2$, on the other hand, encourages PRIORITYNET to maximize its output scores whenever possible, which prevents the system from settling for a trivial solution where both $\mathbf{f}$ and $\mathbf{g}$ are squashed to zero. The trade-off between these two components is controlled by the magnitude of the hyper-parameter λ. Finally, we confirm that this distance metric is fully differentiable with respect to α, β, hence can be efficiently optimized using gradient-based techniques. The parameter gradients are given by:

$$\begin{aligned}\frac{\partial}{\partial\alpha}\Delta_{\mathcal{DM}}(\mathbf{f},\mathbf{g}) &= \sum_{i=1}^{l} a_i \cdot \frac{\partial}{\partial\alpha}\mathbf{f}_i \ , \\ \frac{\partial}{\partial\beta}\Delta_{\mathcal{DM}}(\mathbf{f},\mathbf{g}) &= \sum_{i=1}^{l} b_i \cdot \frac{\partial}{\partial\beta}\mathbf{g}_i \ , \end{aligned} \tag{B.13}$$

where the respective constants are derived as follows:

$$\begin{aligned} a_i &= 2\cdot(1-\mathbf{g}_i)\cdot(\mathbf{f}_i-\mathbf{g}_i) + 2\lambda\cdot(\mathbf{f}_i-1) \ , \\ b_i &= -2\cdot(1-\mathbf{g}_i)\cdot(\mathbf{f}_i-\mathbf{g}_i) - (\mathbf{f}_i-\mathbf{g}_i)^2 \ . \end{aligned} \tag{B.14}$$

B.2 Experiments

We implement our method using PyTorch and deploy all experiments on a RTX-2060 GPU. Similar to many other deep learning workflows, each training epoch computes a batch loss which averages over $N = 10$ randomly sampled subsequences of length $l = 500 \times (w+k)$. We set $\lambda = 1$ and use architectures of PRIORITYNET and TEMPLATENET as given in Fig. B.1 and Section B.1.3 respectively. Network weights are optimized using the ADAM optimizer [46] with learning rate $\eta = 5e^{-3}$.

B.2.1 Comparison baselines

We compare DEEPMINIMIZER with the following benchmarks: (a) random minimizer baseline; (b) Miniception [85]; (c) PASHA [18]; and (d) PolarSet Minimizer [86]. Among these methods, (d) is a sequence-specific minimizer scheme. For each method, we measure the density factor $\mathcal{D}(S;\cdot) \triangleq (w+1)D(S;\cdot)$

to align with the convention of previous work (i.e., the theoretical lower bound on density factor is thus $\mathcal{D}(S;\cdot) \geq 1 + 1/w$). Our empirical result is obtained on different segments of the human reference genome: (a) chromosome 1 (CHR1); (b) chromosome X (CHRX); (c) the centromere region of chromosome X [62] (which we denote by CHRXC); and (d) the full genome (HG38). We used lexicographic ordering for PASHA as suggested by Zheng et al. [85]. Random ordering is used to rank k-mers within the UHS for Miniception, and outside the layered sets for PolarSet. In most settings, we employ the Ensemble template model (Section B.1.3) with no positional phase-shift component (Section B.1.3) for DEEPMINIMIZER. However, for scenarios with large w values, we demonstrate that the Fourier template model with positional phase-shift is able to achieve better performance (Section B.2.7)

B.2.2 Visualizing the mechanism of DEEPMINIMIZER

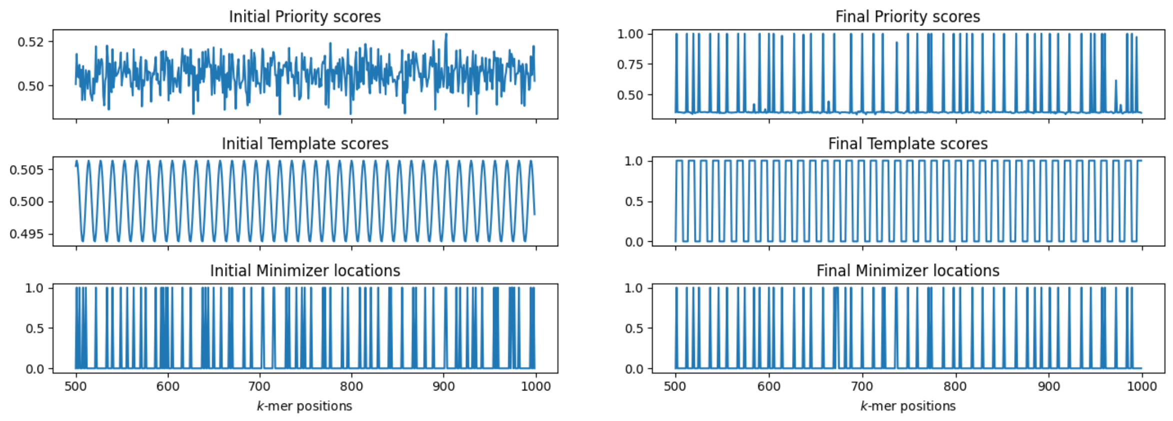


Figure B.2: Visualization of PRIORITYNET and TEMPLATENET score assignments on positions $500 - 1000$ of CHRXC with $w = 13$, $k = 8$. Left: Initial assignments ($\mathcal{D} = 2.05$); Right: Final assignments after 600 training epochs ($\mathcal{D} = 1.39$). The bottom plots show corresponding locations of sampled k-mers: a value of 1 means selected, and 0 otherwise.

First, we show the transformation of the priority scores assigned by SCORENET and TEMPLATENET over 600 training epochs. Fig. B.2 plots the outputs of these networks evaluated on positions 500 to 1000 of CHRXC, and their corresponding locations of sampled k-mers.

For ease of implementation, we employ the standard MAXPOOL operator from PyTorch to select window maxima as minimizer locations (instead of window minima, as previously formulated) . As a result, we expect the sampled locations in Fig. B.2 to coincide with the peaks of the priority scores (instead of the troughs). We also note that to accommodate this implementation, every relevant term in the DEEPMINIMIZER objective has been properly negated.

Initially, the PRIORITYNET assignment resembles that of a random minimizer and expectedly yields $\mathcal{D} = 2.05$. After 600 training epochs, the final TEMPLATENET assignment converges with a different phase shift than its initial assignment, but its period remains the same. Simultaneously, PRIORITYNET learns to match this template, hence induces a visibly sparser sketch with $\mathcal{D} = 1.39$. This result demonstrates the negotiating behaviour of our twin architecture to find an optimal consensus score assignments.

B.2.3 Convergence of our proxy objective

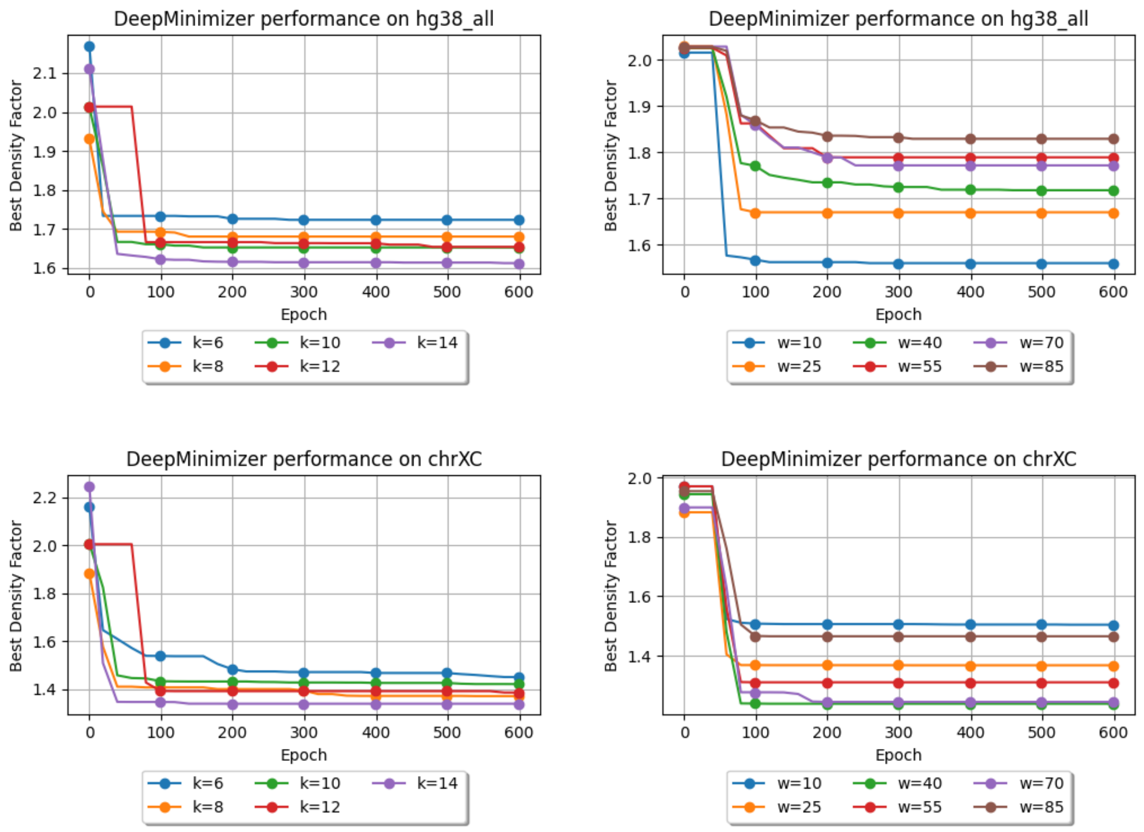


Figure B.3: Best density factors obtained by DEEPMINIMIZER on HG38, CHRXC over 600 training epochs. Left: fix $w = 13$, and vary $k \in \{6, 8, 10, 12, 14\}$; Right: fix $k = 14$, and vary $w \in \{10, 25, 40, 55, 70, 85\}$.

We further demonstrate that our proxy objective meaningfully improves minimizer performance as it is optimized. The first two columns of Fig. B.3 show the best density factors achieved by our method over 600 epochs on two scenarios: (a) varying k with fixed w; and (b) varying w with fixed k. The experiment is repeated on CHRXC and HG38. In every scenario, DEEPMINIMIZER starts with $\mathcal{D} \simeq 2.0$, which is only comparable to a random minimizer. We observe steady decrease of $\mathcal{D}$ over the first 300 epochs before reaching convergence, where total reduction ranges from $11 - 23\%$.

Generally, larger k values lead to better performance improvement at convergence. This is expected since longer k-mers are more likely to occur uniquely in the target sequence, which makes it easier for a minimizer to achieve sparse sampling. In fact, previous results have shown that when k is much smaller than $\log w$, no minimizer will be able to achieve the theoretical lower-bound $\mathcal{D}$ [85]. On the other hand, larger w values lead to smaller improvements and generally slower convergence. This is because our ensemble parameterization of TEMPLATENET scales with the window size w and becomes more complicated to optimize as w increases.

B.2.4 Evaluating our proposed distance metric

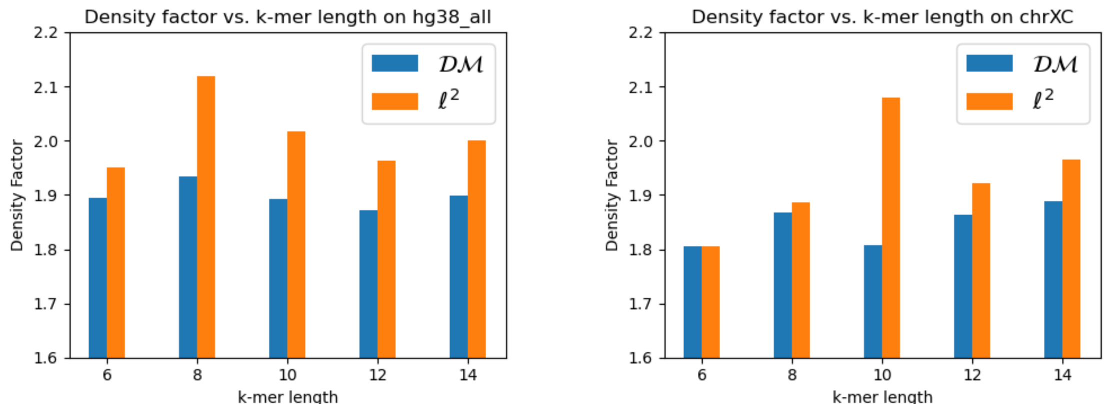


Figure B.4: Comparing best density factors obtained by DEEPMINIMIZER with Δ_{ℓ^2} and $\Delta_{\mathcal{DM}}$ on HG38 (left) and CHRXC (right) over 600 training epochs.

Fig. B.4 shows the density factors achieved by our DEEPMINIMIZER method, respectively specified by the proposed distance metric $\Delta_{\mathcal{DM}}$ in Eq. B.12 and Δ_{ℓ^2} distance. Here, we fix $w = 13$ and vary $k \in \{6, 8, 10, 12, 14\}$. We observe that with the Δ_{ℓ^2} distance, we obtain performance similar to a random minimizer in most cases. On the other hand, with our divergence function, DEEPMINIMIZER obtains significantly lower densities, which confirms the intuition in Section B.1.4.

B.2.5 Comparing against other minimizer methods

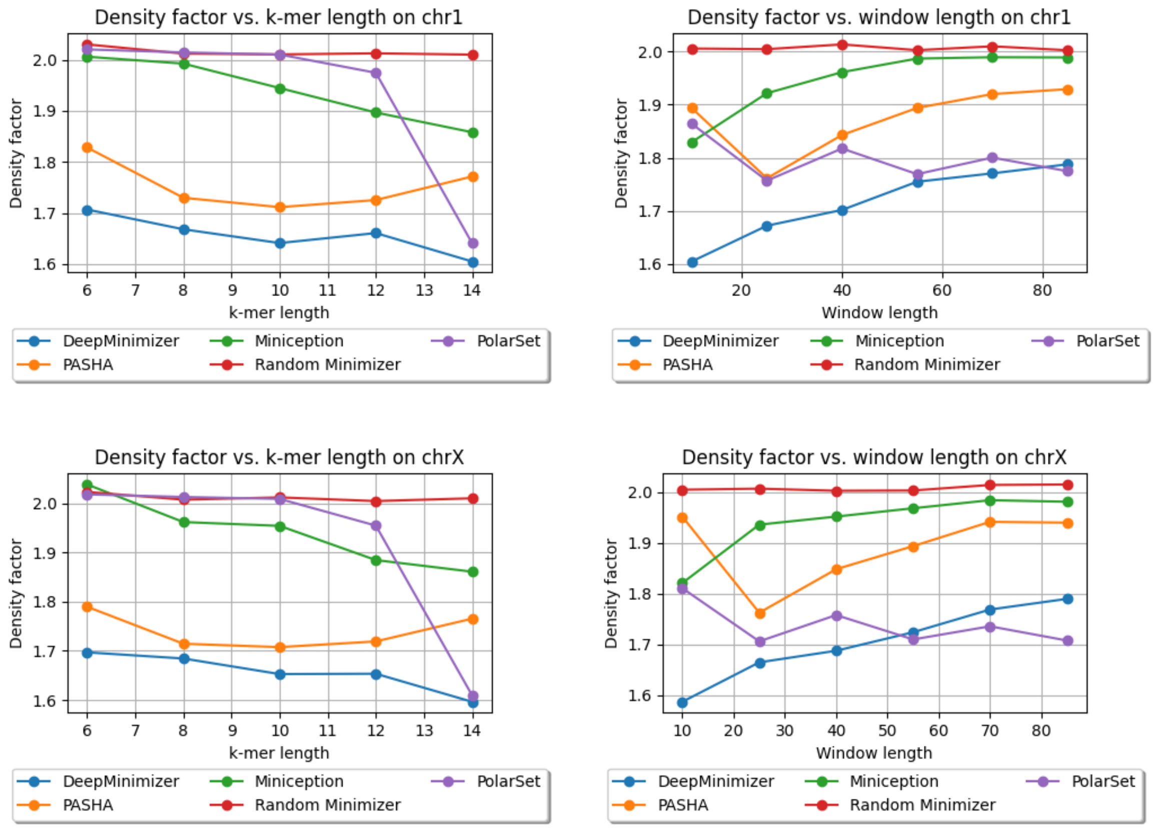


Figure B.5: Density factors obtained by DEEPMINIMIZER (600 training epochs), Random Minimizer, PASHA, Miniception and PolarSet on CHR1, CHRX. Left: fix $w = 13$, and vary $k \in \{6, 8, 10, 12, 14\}$; Right: fix $k = 14$, and vary $w \in \{10, 25, 40, 55, 70, 85\}$.

We show the performance of DEEPMINIMIZER compared to other benchmark methods. In this experiment, DEEPMINIMIZER is trained for 600 epochs with ensemble TEMPLATENET and no positional phase-shift. Fig. B.5 and Fig. B.6 shows the final density factors achieved by all methods, again on two comparison scenarios: (a) fix $w = 13$, and vary $k \in \{6, 8, 10, 12, 14\}$; and (b) fix $k = 14$, and vary $w \in \{10, 25, 40, 55, 70, 85\}$. DEEPMINIMIZER consistently achieves better performance compared to *non-sequence-specific* minimizers (i.e., PASHA, Miniception) on all settings. We observe up to 40%

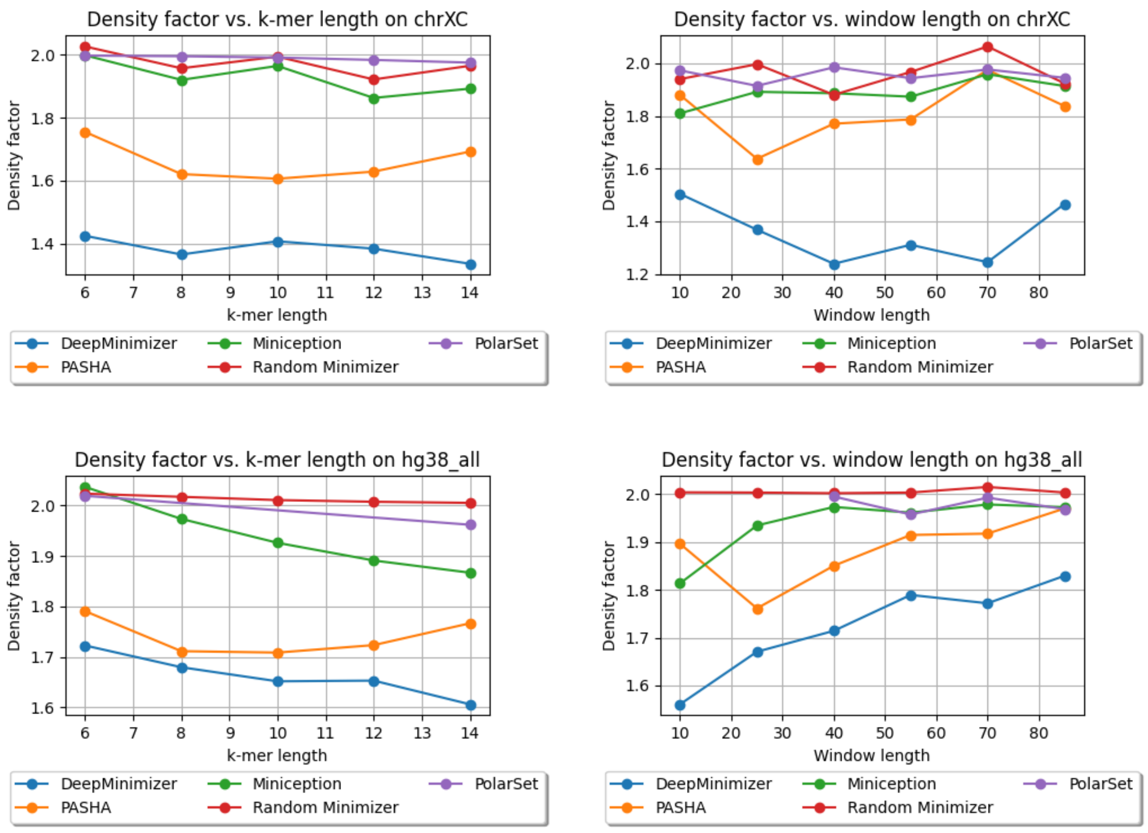


Figure B.6: Density factors obtained by DEEPMINIMIZER (600 training epochs), Random Minimizer, PASHA, Miniception and PolarSet on CHRXC, HG38. Left: fix $w = 13$, and vary $k \in \{6, 8, 10, 12, 14\}$; Right: fix $k = 14$, and vary $w \in \{10, 25, 40, 55, 70, 85\}$.

reduction of density factor (e.g., on CHRXC, $w = 70$, $k = 14$), which clearly demonstrates the ability of DEEPMINIMIZER to exploit *sequence-specific* information. Furthermore, we also observe that DEEPMINIMIZER outperforms our *sequence-specific* competitor, PolarSet, in a majority of settings. The improvements over PolarSet are especially pronounced for smaller k values, which are known harder tasks for minimizers [85]. On larger w values, our method performs slightly worse than PolarSet in some settings. This is likely due to the added complexity of optimizing TEMPLATENET, as described in convergence ablation study of our method.

Notably, the centromere region of chromosome X (i.e., CHRXC) contains highly repetitive subsequences [24] and has been shown to hamper performance of PolarSet [86]. Fig. B.6 shows that PolarSet and the UHS-based methods perform similarly to a random minimizer, whereas our method is consistently better. Moreover, we observe that DEEPMINIMIZER obtains near-optimal densities with CHRXC on several settings. For example, we achieved $\mathcal{D} = 1.22$ when $k = 14$, $w \in \{40, 70\}$, which is significantly better than the results on CHR1 and CHRX. This suggests that CHRXC is not necessarily more difficult to sketch, but rather good sketches have been excluded by the UHS and polar set reparameterizations, which is not the case with our framework.

B.2.6 Number of unique k-mers in the final minimizer set

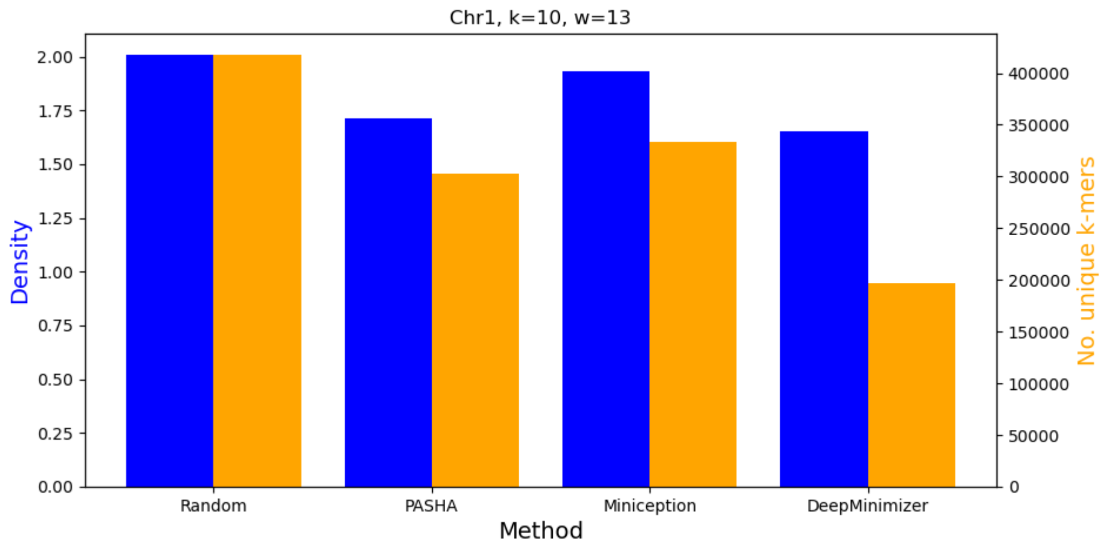


Figure B.7: Comparing density and number of unique k-mers in the minimizer sets obtained by various benchmarks on CHR1 with $k = 10$ and $w = 13$.

This section investigates the numbers of unique k-mers in the final minimizer sets obtained by random ordering, PASHA, Miniception and DeepMinimizer. On Chromosome 1, with $k = 10$ and $w = 13$, Fig B.7 shows that the density factors and numbers of unique k-mers obtained by each method are strongly correlated. This agrees with the intuition of many other minimizer methods that a small set of high priority k-mers (e.g., a small UHS in the case of PASHA and Miniception) tends to induce a low density sketch on the target sequence. This observation is also expected since the 10-mer distribution of CHR1 is fairly similar to that of a random sequence, which aligns with the premise of most UHS-based minimizer theories.

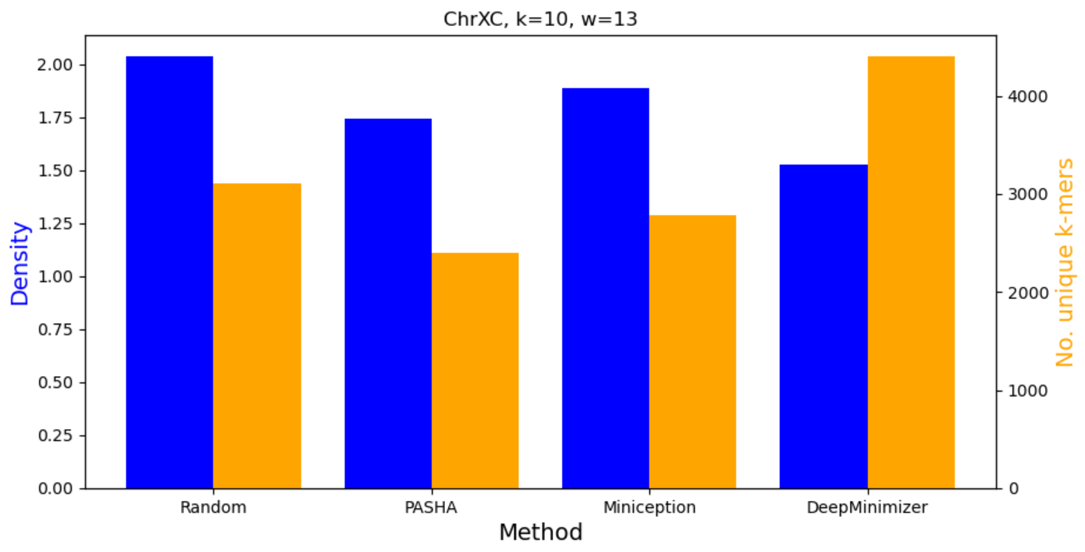


Figure B.8: Comparing density and number of unique k-mers in the minimizer sets obtained by various benchmarks on CHRXC with $k = 10$ and $w = 13$.

However, on the chromosome region of CHRX, which contains many highly repetitive sub-sequences, Fig. B.8 shows that in order to achieve the best density (i.e., $\mathcal{D} = 1.526$), DEEPMINIMIZER actually had to pick more high priority k-mers, not fewer. This interestingly demonstrates that minimizing the size of the UHS is not always a desirable surrogate objective on certain specific sequences, hence asserts the need for a robust sequence-specific optimizer.

B.2.7 Comparing template models on large window values

In this section, we investigate the performance of DEEPMINIMIZER on large window size with different template models. Particularly, we fixed $k = 20, w = 100$ and compare the best density factor obtained by DEEPMINIMIZER over 1200 training epochs using the ensemble template model (Section B.1.3) and the truncated Fourier series template model (Section B.1.3). We further pair each template model with a positional phase-shift component (Section B.1.3), with $\epsilon \in \{0.0, 1.0, 10.0\}$. We note that in each case, $\epsilon = 0.0$ corresponds to the original template model.

Fig. B.9 shows the respective loss and density factor over 1200 training epochs of these template models. First, we observe that in all models, the loss values correlate positively with the corresponding

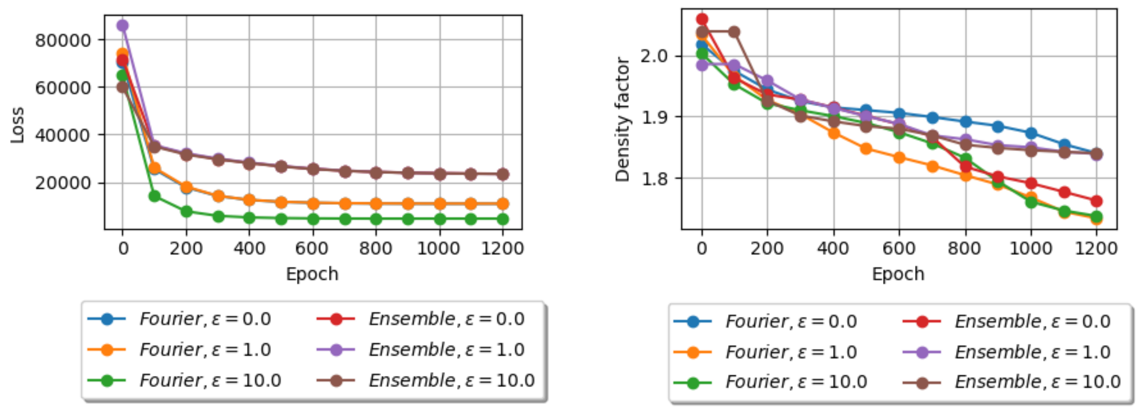


Figure B.9: Comparing loss (left) and best density obtained (right) over 1200 training epochs on CHR1 between ensemble and truncated Fourier series template models. Each template model is paired with a positional phase-shift component with $\epsilon \in \{0.0, 1.0, 10.0\}$.

density factor. Generally, as the DEEPMINIMIZER loss decreases, the induced minimizer scheme also yields lower density factor on the input sequence, which suggests that our loss function is a good surrogate for the discrete density objective.

Furthermore, we observe that among variants of the Fourier template model, both $\epsilon = 1.0$ and $\epsilon = 10.0$ perform significantly better than $\epsilon = 0.0$. This is most likely because adding local phase perturbations indeed allows TEMPLATENET to encode more realistic near-optimal score assignments. In contrast, among variants of the ensemble template model, $\epsilon = 0.0$ performs the best. This is most likely because the ensemble model has already accounted for all possible integer phase-shifts. As such, adding noisy phase perturbations with magnitude greater than 1.0 will negatively affect the convergence of DEEPMINIMIZER.

Finally, pairing Fourier template model with a positional phase-shift component of magnitude $\epsilon = 1.0$ achieves the best performance out of all variants. This aligns with our intuition in Section B.1.3 regarding the trade-off between the certainty of Proposition 1 and the expressiveness of the admitted set of template score assignments.

B.2.8 Runtime performance

Finally, we confirm that DEEPMINIMIZER runs efficiently with GPU computing. In all of our experiments, each training epoch takes approximately 30 seconds to 2 minutes, depending on the choice of k and w, which controls the batch size. Performance evaluation takes between several minutes (CHRXC) to 1 hour (HG38), depending on the length of the target sequence. Generally, our method is cost-efficient without frequent evaluations. Our most cost-intensive experiment (i.e., convergence ablation study on HG38) requires a full-sequence evaluation every 20 epochs over 600 epochs, thus takes approximately 2 days to complete. This is faster than PolarSet, which has a theoretical runtime of $\mathcal{O}(n^2)$ and takes several days to run with HG38. We note that in real applications, we only have to evaluate once by the end of the training loop, which is much faster compared to PolarSet, whose running time above only involves building the minimizer scheme.

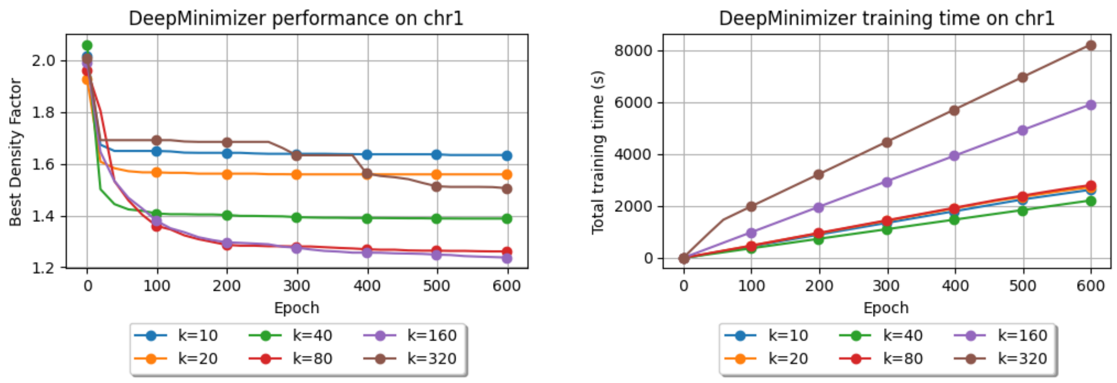


Figure B.10: Best density obtained (left) and runtime (right) of DEEPMINIMIZER for $w = 13$ and $k \in \{10, 20, 40, 80, 160, 320\}$ on CHR1.

Fig. B.10 (right) measures runtime (in seconds) of DEEPMINIMIZER on CHR1 over 600 epochs. Larger k values require PRIORITYNET to have more parameters. We expect running time for $k = 40, 80, 160, 320$ to increase in the same order. For $k = 10$ and 20, however, the running times are approximately the same as $k = 80$. We note that a smaller k value means there are more k-mers in the same sequence. As such, even though PRIORITYNET is more compact for these values of k, we will incur some overhead from querying it more often. For completeness, we also show the corresponding density performance plot in Fig. B.10 (left), which confirms that our model converges well even for large k.

Appendix C

Masked Minimizer Appendix

C.1 Comparing and Unifying Sketching Schemes

C.1.1 Revisiting performance metrics

Let $(\mathcal{X}, r)$ be an arbitrary sampling scheme. As the set of conserved locations $\mathcal{X}(S) \cap \mathcal{X}(S')$ is always a subset of $\mathcal{X}(S)$ regardless of the mutations in S', we have $\mathbb{E}_{S'}\left[|\mathcal{X}(S) \cap \mathcal{X}(S')|\right] \leq |\mathcal{X}(S)|$, and therefore $\mathcal{C}(S; \mathcal{X}, r) \leq \mathcal{D}(S; \mathcal{X}, r)$ for all S. Improving (lowering) density thus places an upper-bound on how much conservation can be improved and vice versa. This conflicting nature between the two objectives implies that neither should be considered independently of one another. To account for this trade-off, we propose a new metric called the *generalized sketch score* (GSS), which is defined as follows:

$$\begin{aligned} G(S; \mathcal{X}, r, w) &\triangleq \frac{C(S; \mathcal{X}, r)}{D(S; \mathcal{X}, r)} \cdot \frac{1}{L_w} \sum_{i=1}^{L_w} V_i(S; \mathcal{X}; w) \\ &= \frac{C(S; \mathcal{X}, r)}{D(S; \mathcal{X}, r)} \cdot \frac{1}{L_w} \sum_{i=1}^{L_w} \left(1 - \prod_{j=i}^{i+w-1} \mathbb{I}(j \notin \mathcal{X}(S))\right) , \end{aligned} \tag{C.1}$$

where $V_i(S; \mathcal{X}, w)$ is the indicator variable of the event $\kappa_i^{w_k}$ overlaps at least one sampled index in $\mathcal{X}(S)$.

Explicitly, our metric consists of two components. The first component, $C(S; \mathcal{X}, r)/D(S; \mathcal{X}, r)$, evaluates the *relative conservation* and captures the inherent trade-off between the density and conservation metrics. Interestingly, this term also corresponds to measuring the number of conserved locations in

$\mathcal{X}(S) \cap \mathcal{X}(S')$ relative to the number of sampled locations in the original sketch $\mathcal{X}(S)$:

$$\begin{aligned}\frac{C(S;\mathcal{X},r)}{D(S;\mathcal{X},r)} &= \frac{\mathbb{E}_{S'}\left[\left|\mathcal{X}(S) \cap \mathcal{X}(S')\right|\right]}{L} \cdot \frac{L}{|\mathcal{X}(S)|} \\ &= \frac{\mathbb{E}_{S'}\left[\left|\mathcal{X}(S) \cap \mathcal{X}(S')\right]\right.}{|\mathcal{X}(S)|}\,. \end{aligned} \tag{C.2}$$

As a metric, this term alone is, however, vulnerable to a simple exploit that trivially maximizes it. To see this, we first note that around each sampled location, there is a finite-length substring (i.e., context) in which a mutation can possibly alter the sampling outcome. With respect to the default reporting function, the context of a minimizer-sampled location is the union of all windows that contain it. The context of a syncmer-sampled location is the k-mer at the same position. Independent of the random mutations, the portion of the sequence in which mutations might induce an effect on the relative conservation metric is therefore bounded by the number of selected locations in $\mathcal{X}(S)$. The fewer locations sampled by $\mathcal{X}$ means a smaller probability for any mutation to occur in this conservation-sensitive portion of the sequence.

Naturally, if the sketch picks an unreasonably small number of locations, it is likely that the relative conservation term will be near perfect (i.e., close to 1). One such scenario could theoretically occur with the syncmer setting, when $f_\pi(\kappa_i^s)$ is constructed such that the lowest scoring s-mer in every k-mer is always found within the first $t-1$ positions. When this happens, a large portion of the sequence is not represented by any k-mer, thus resulting in a meaningless sketch. We demonstrate that such a sampling behavior can be found via optimization in Section C.3. To prevent this sub-optimal outcome, our second term $\sum_{i=1}^{L_w} V_i(S,\mathcal{X};w)/L_w$ measures the *coverage* of the sketch, or the fraction of (w,k)-windows that contains at least one sampled k-mer. When very few k-mers are selected, the resulting low coverage will apply a discount on the high relative conservation term, hence will discourage these trivial solutions.

C.1.2 Comparable minimizers and syncmers

Minimizer and syncmer schemes with similar k are typically deemed comparable [16, 74] since they both adopt the k-mer reporting function $r(i) = (\kappa_i^k, i)$. This notion of comparability, however, does not facilitate a theoretical analysis of their performance differences (i.e., in terms of density and conservation metrics), as different information bases are used to enact the sampling decisions of a length-k minimizer and syncmer. In particular, (w,k)-minimizer schemes use total k-mer orderings to perform sampling, whereas (k, s, t)-syncmer schemes use total s-mer orderings, with $s \leq k$. These bases are only comparable when we set $s = k$, but doing so results in trivial a scheme that selects every k-mer in S.

To correct this asymmetry of information, we propose to compare π-comparable minimizers and syncmers, which are sketching schemes that employ the same total ordering π. For example, the (w, k, π)-minimizer and the (w_k, k, t, π)-syncmer schemes with $w_k \triangleq w + k - 1$ and $t \leq w$, which respectively report k-mers and w_k-mers, are π-comparable. To work around their difference in representation, we further replace the default w_k-mer reporting function $r(i) = (\kappa_i^{w_k}, i)$ of the above syncmer scheme with the k-syncmer reporting function $r'(i) = (\kappa_{i+t}^{k}, i + t)$. As both functions are one-to-one mappings of the selected locations, this substitution results in a semantically equivalent *shifted* syncmer scheme that reports all k-mers that are both lowest ranked and at the t^{th} position in their respective w_k-mers.

This translation of the reporting function aligns our proposed comparison with the traditional perspective of comparability and presents an invariant substring ranking behavior among the compared schemes, which subsequently allows us to derive explicit bounds on their density and conservation gaps. In particular, Proposition 2 proves that the sketch of any syncmer is a subset of its π-comparable minimizer sketch. Corollary C.1.0.1 and Corollary C.1.0.2 further show that the density and conservation of any syncmer on S are respectively upper-bounded and almost surely upper-bounded by that of its π-comparable minimizer, thus establishing the first theoretical correspondence between π-comparable schemes.

Proposition 2 *Given $w, k \in \mathbb{N}$ and a total ordering π defined on the set of all k-mers, we let (w, k, π) and the reporting function $r(i) = (\kappa_i^k, i)$ define a minimizer scheme $(\mathcal{M}, r)$. Further let (w_k, k, t, π) and $r'(i) \triangleq (\kappa_{i+t}^k, i+t)$ define a shifted syncmer scheme $(\mathcal{O}_t, r')$, such that $t \leq w$. Then, for all $S \in \Sigma^{L+k-1}$, we have $\mathcal{K}(S; \mathcal{O}_t, r') \subseteq \mathcal{K}(S; \mathcal{M}, r)$.*

Proof: *We first note that $\mathcal{O}_t$ will not sample any location i such that κ_{i+t}^k, or the t^{th} k-mer in $\kappa_i^{w_k}$, does not exist, hence r' is well-defined. Then, by definition of r and r', it suffices to show that $i \in \mathcal{O}_t(S) \Rightarrow i + t \in \mathcal{M}(S)$ for all $i \in [L_w]$. As both schemes are parameterized by the same ordering π, we can express their respective sets of sampled locations using the same selector function m_π:*

$$\mathcal{M}(S) = \{i + m_\pi(\kappa_i^{w_k})\}_{i \in [L_w]} \quad \textit{and} \quad \mathcal{O}_t(S) \ = \ \{i \mid m_\pi(\kappa_i^{w_k}) = t\}_{i \in [L_w]}. \tag{C.3}$$

Therefore, for all $i \in [L_w]$, we have $i \in \mathcal{O}_t(S) \Rightarrow i + t = i + m_\pi(\kappa_i^{w_k}) \in \mathcal{M}(S)$. □

Corollary C.1.0.1 (Density gap of π-comparable schemes) *Let (w_k, k, t, π, r') and (w, k, π, r) define a pair of π-comparable shifted syncmer $(\mathcal{O}_t, r')$ and minimizer $(\mathcal{M}, r)$ schemes as described in Proposition 2, then for all $S \in \Sigma^{L+k-1}$, we have $D(S; \mathcal{O}_t, r') \leq D(S; \mathcal{M}, r)$.*

Proof: *This follows directly from Proposition 2, which establishes that $\mathcal{K}(S; \mathcal{O}_t, r') \subseteq \mathcal{K}(S; \mathcal{M}, r)$. Thus, we have $D(S; \mathcal{O}_t, r') = |\mathcal{K}(S; \mathcal{O}_t, r')|/L \leq |\mathcal{K}(S; \mathcal{M}, r)|/L = D(S; \mathcal{M}, r)$.* $\square$

Corollary C.1.0.2 (Conservation gap of π-comparable schemes) *Let (w_k, k, t, π, r') and (w, k, π, r) define a pair of π-comparable shifted syncmer $(\mathcal{O}_t, r')$ and minimizer $(\mathcal{M}, r)$ schemes as described in Proposition 2, then for all $S \in \Sigma^{L+k-1}$, we have $C(S; \mathcal{O}_t, r') \leq C(S; \mathcal{M}, r) + t/L$.*

Proof: *Fixing a mutation S' and let $\mathcal{X}$ be an arbitrary location sampling function, we additionally define the indicator variable $\alpha_i(S', \mathcal{X}) \triangleq \mathbb{I}(i \in \mathcal{X}(S) \cap \mathcal{X}(S'))$ of the event that location i is preserved across $\mathcal{X}(S)$ and $\mathcal{X}(S')$. We first give the following bound of $\alpha_i(S', \mathcal{O}_t)$ in terms of $\alpha_{i+t}(S', \mathcal{M})$:*

$$\begin{aligned} \alpha_i(S', \mathcal{O}_t) &= \mathbb{I}(i \in \mathcal{O}_t(S)) \times \mathbb{I}(i \in \mathcal{O}_t(S')) \\ &\leq \mathbb{I}((i+t) \in \mathcal{M}(S)) \times \mathbb{I}((i+t) \in \mathcal{M}(S')) \\ &= \alpha_{i+t}(S', \mathcal{M}) \,, \end{aligned} \tag{C.4}$$

where the inequality follows from Proposition 2, which establishes that $i \in \mathcal{O}_t(S) \Rightarrow (i+t) \in \mathcal{M}(S)$. As r and r' are both one-to-one mappings, we rewrite the conservation difference between $\mathcal{O}_t$ and $\mathcal{M}$ as:

$$C(S; \mathcal{O}_t, r') - C(S; \mathcal{M}, r) = \mathbb{E}_{S'}\left[\frac{|\mathcal{O}_t(S) \cap \mathcal{O}_t(S')| - |\mathcal{M}(S) \cap \mathcal{M}(S')|}{L}\right] \tag{C.5}$$

Expressing the cardinality of each intersection set above as the sum of its indicator variables gives:

$$
\begin{aligned}
C(S;\mathcal{O}_t,r') - C(S;\mathcal{M},r) &= \mathbb{E}_{S'}\left[\sum_{i=1}^{L}\frac{\alpha_i(S',\mathcal{O}_t)-\alpha_i(S',\mathcal{M})}{L}\right] \\
&= \mathbb{E}_{S'}\left[\sum_{i=1}^{L-t}\frac{\alpha_i(S',\mathcal{O}_t)-\alpha_{i+t}(S',\mathcal{M})}{L}\right] \\
&+ \mathbb{E}_{S'}\left[\sum_{i=1}^{t}\frac{\alpha_{L-i+1}(S',\mathcal{O}_t)-\alpha_i(S',\mathcal{M})}{L}\right] \\
&\leq \mathbb{E}_{S'}\left[\sum_{i=1}^{t}\frac{\alpha_{L-i+1}(S',\mathcal{O}_t)-\alpha_i(S',\mathcal{M})}{L}\right] \\
&\leq \mathbb{E}_{S'}\left[\sum_{i=1}^{t}\frac{1}{L}\right] = \frac{t}{L}\,, \qquad \text{(C.6)}
\end{aligned}
$$

where the first inequality follows from the bound in Eq. (C.4) *and the second inequality is due to the fact that each α term is either* 0 *or* 1*. Rearranging the above result concludes our proof.* □

We note that since $t \leq k \ll L$ in most practical applications, Corollary C.1.0.2 almost surely corresponds to an exact upper bound. Finally, we state our main result in Theorem C.1.1 below, which establishes the first theoretical correspondence between minimizers and syncmers. That is, for every syncmer, there exists a corresponding minimizer scheme that simultaneously yields higher (worse) density and higher (better) conservation.

Theorem C.1.1 (π-comparability defines a correspondence between minimizers and syncmers) *For every syncmer scheme $(\mathcal{O}_t, r')$, there exists a comparable minimizer scheme $\mathcal{M}$ whose reporting function r is translated from r' with an offset t, such that the following bounds hold for every $S \in \Sigma^{L+k-1}$:*

$$D(S;\mathcal{O}_t,r') \leq D(S;\mathcal{M},r) \qquad \textit{and} \qquad C(S;\mathcal{O}_t,r') \leq C(S;\mathcal{M},r) + \frac{t}{L}\,. \qquad \text{(C.7)}$$

Proof: *The proof of Theorem C.1.1 follows directly from Corollary C.1.0.1 and Corollary C.1.0.2.* □

C.1.3 Unifying comparable schemes with masked minimizers

Proposition 2 establishes that $\mathcal{K}(S;\mathcal{O}_t,r') \subseteq \mathcal{K}(S;\mathcal{M},r)$ for any π-comparable pair of schemes $(\mathcal{O}_t,r')$ and $(\mathcal{M},r)$. Additionally, it follows from the definitions of $\mathcal{O}_t$ and $\mathcal{M}$ that, for all $i \in [L]$, $m_\pi(\kappa^{w_k}_{i-t}) = t$ simultaneously implies $i \in \mathcal{M}(S)$ and $i - t \in \mathcal{O}_t(S)$. Thus, $m_\pi(\kappa^{w_k}_{i-t}) = t$ suffices as a sub-sampling rule to recover $\mathcal{O}_t(S)$ given $\mathcal{M}(S)$, or to recover $\mathcal{K}(S;\mathcal{O}_t, r')$ given $\mathcal{K}(S;\mathcal{M}, r)$. We write this rule as:

$$\mathcal{O}_t(S) = \{i - t \mid m_\pi(\kappa^{w_k}_{i-t}) = t\}_{i\in\mathcal{M}(S)} \,. \tag{C.8}$$

Let the output of the selector function m_π be equivalently represented as a w-dimensional one-hot vector $\overrightarrow{m}_\pi(\kappa^{w_k}) \triangleq [\mathbb{I}(j = m_\pi(\kappa^{w_k}))]_{j\in[w]}$, where κ^{w_k} is some arbitrary (w, k)-window in S. The above sub-sampling rule can then be rewritten as the point-wise multiplication between $\overrightarrow{m}_\pi(\kappa^{w_k}_{i-t})$ and the w-dimensional one-hot vector $\mathbf{e}_t = [\mathbb{I}(j = t)]_{j\in[w]}$, whose 1-entry is at the t^{th} position:

$$\mathcal{O}_t(S) = \left\{i - t \mid \left\|\overrightarrow{m}_\pi(\kappa^{w_k}_{i-t}) \otimes \mathbf{e}_t\right\|_1 > 0\right\}_{i\in\mathcal{M}(S)} \,. \tag{C.9}$$

In the above formulation, the positive 1-norm condition checks if any position remains selected after filtering with $\mathbf{e}_t$ (in which case, it must be the t^{th} position). Interestingly, this sub-sampling rule can be generalized by replacing $\mathbf{e}_t$ with any arbitrary w-dimensional binary mask $\nu \in \{0, 1\}^w$. For example, setting $\nu = \mathbf{1}_w$ trivially recovers the standard minimizer scheme (without applying the offset t). Given a set of minimizer-sampled locations $\mathcal{M}(S)$, varying ν yields a total of 2^w distinct π-comparable schemes, leading to a unifying method called *masked minimizers*. We state its definition below.

Masked minimizers.

The sampling function of a masked minimizer scheme is characterized by a tuple of parameters (w, k, π, ν), where w, k, π correspond to standard minimizer parameters, and $\nu \in \{0, 1\}^w$ is a w-dimensional binary vector. The masked minimizer sampling function is given by:

$$\mathcal{V}(S; w, k, \pi, \nu) \;\triangleq\; \{i + m_\pi(\kappa^{w_k}_i) \mid \zeta(\kappa^{w_k}_i, \nu)\}_{i\in[L_w]} \,, \tag{C.10}$$

where $\zeta(\kappa^{w_k}_i, \nu) \triangleq \|\overrightarrow{m}_\pi(\kappa^{w_k}_i) \otimes \nu]\|_1 > 0$ denotes the event that the selection at the i^{th} window remains sampled after applying the sub-sampling mask. We note that this definition does not factor in the offset t to recover exactly the locations sampled by a comparable syncmers. We can, however, specify the reporting function $r(i) = (\kappa^{w_k}_{i-t}, i - t)$ to recover $\mathcal{O}_t(S; w_k, k, \pi)$ from $\mathcal{V}(S)$. Thus, every syncmer and minimizer scheme can be written as a masked minimizer.

C.2 Optimizing masked minimizers

Following the standard setting of minimizer optimization, we fix w, k for the masked minimizer scheme and optimize for π, ν. We address this objective via a bi-level optimization routine, which first optimizes for π, then performs greedy search for the optimal sub-sampling mask ν, which starts with the minimizer mask $\mathbf{1}_w$ and iteratively zeroes out a position that yields the best performance improvement. The search terminates when no further improvement can be obtained. In principle, any existing optimizer for minimizer [18, 31, 85, 86] can be used in the first step. However, we note that all existing algorithms to optimize minimizers do not account for the conservation component that is reflected in the GSS metric. To address this, we adapt the DEEPMINIMIZER loss function [31] with a secondary objective that minimizes the expected change in k-mer score assignment with respect to random mutations, hence encouraging high conservation. Our adapted loss function to optimize GSS is given as follows:

$$\mathcal{L}_{gss}(S; \alpha, \beta) \quad \triangleq \quad \Delta(\mathbf{f}(S; \alpha), \mathbf{g}(S; \beta)) + \sum_{i=1}^{n} \Delta(\mathbf{f}(S_i; \alpha), \mathbf{g}(S; \beta)) \,, \tag{C.11}$$

where $S_1, S_2, \ldots, S_n$ denote n randomly sampled mutations of S. The first term on the right hand side is exactly the density optimizing DEEPMINIMIZER loss [31]. The second term minimizes the expected distance between each priority vector $\mathbf{f}(S_i; \alpha)$ (of the mutated sequence S_i) and the template vector $\mathbf{g}(S; \beta)$ (of the original sequence). Intuitively, when this term is small, we expect all $\mathbf{f}(S_i; \alpha)$ to be concentrated around $\mathbf{g}(S; \beta)$. As $\mathbf{g}(S; \alpha)$ is close to $\mathbf{f}(S; \alpha)$ via minimizing the first term, this optimality also implies that the score assignment $\mathbf{f}(S; \alpha)$ is likely to be conserved under random mutations.

C.3 Empirical results

Our experiments aim to investigate the following: (1) Are density and conservation adversarially related? (2) How do π-comparable schemes perform relative to one another under the proposed metric GSS? (3) Can mask optimization improve the overall performance of both minimizer and syncmer? Through these demonstrations, we confirm our theoretical understanding of various sketching metrics and the relationship among π-comparable schemes, as well as demonstrate the efficiency of our proposed optimization method. We further conduct ablation studies to investigate several interesting phenomena related to masked minimizers, such as the ability to exploit the relative ordering metric (as mentioned in Section C.1.1); the performance of all masks, which reveals insight on setting a good default mask; and the ability to prevent repeated sampling in a homopolymer-rich sequence, which has previously been a unique advantage of the syncmer method.

Experimentation details.

We compare the following methods to construct the k-mer ordering: (1) random ordering; (2) training with different variants of the DEEPMINIMIZER loss [31]; (3) constructing the ordering from MINICEPTION [85]; and (4) PASHA universal hitting sets [18]. Performance is demonstrated on the following mask parameters: (1) the minimizer mask $\nu = \mathbf{1}_w$; (2) the syncmer mask with offset $t = w/2$ (suggested by Shaw and Yu [74]) meaning $\nu = \mathbf{e}_{w/2}$; (3) its complement $\nu = \mathbf{1}_w - \mathbf{e}_{w/2}$; and (4) the optimized mask ν_* found by the heuristic search described above. We respectively label the sketches of these masked minimizers as $\mathcal{M}(S), \mathcal{O}_{w/2}(S), \mathcal{C}_{w/2}(S)$ and $\mathcal{V}(S)$. All experiments are trained on human chromosome 1 (labelled CHR1); the centromere region of human chromosome X (labelled CHRXC); and several bacterial genomes that were previously used in Edgar [16] (labelled BTR1, BTR2, BTR3 and BTR4). The details of these sequences are given in Appendix C.4. We compute all gradient-based loss functions per batch of sampled subsequences since it is not possible to fit the entire sequence on GPU memory. Other implementation details are given in Appendix C.5. Our implementation can be found at `https://github.com/hqminh/maskedminimizer`.

The adversarial relationship of density and conservation.

This experiment demonstrates that density and conservation are conflicting objectives for π-comparable schemes, thus confirming our analysis in Section C.1.1. First, we train masked minimizers for $w = 7, k = 15$ and various binary masks ν using three different loss functions: (1) the DEEPMINIMIZER loss, which only optimizes for density and is labelled as $\mathcal{L}_{DM}$; (2) only the conservation term in Eq. (C.11), which optimizes for conservation and is labelled as $\mathcal{L}_{con}$; and (3) the combined masked minimizer loss, which is given as Eq. (C.11) and labelled as $\mathcal{L}_{gss}$.

Fig. C.1 plots the density, conservation, coverage, relative conservation and GSS metrics as these

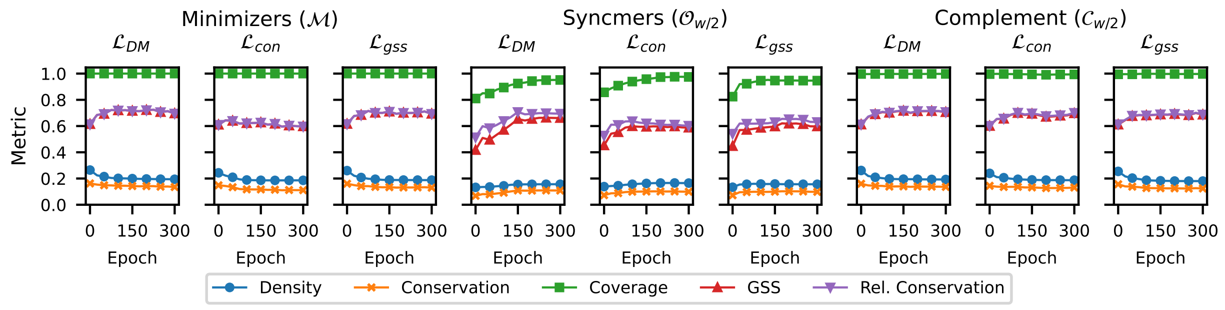
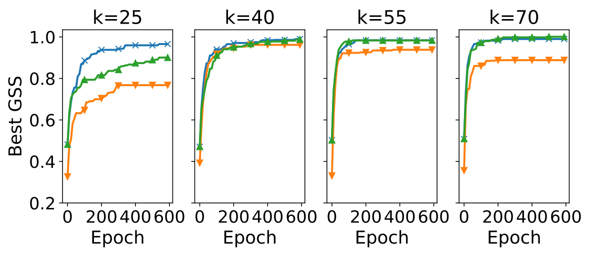


Figure C.1: Comparing density, conservation, relative conservation and GSS vs. number of training epochs using difference training losses and masks on the bacterial genome BTR1.

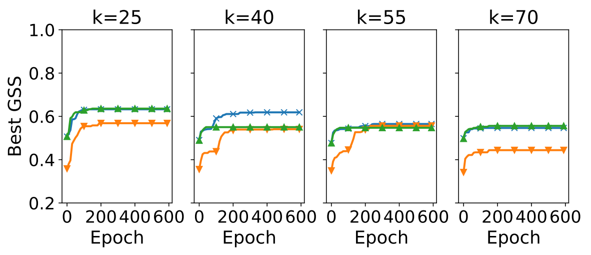
masked minimizer schemes are trained with each loss function over 300 epochs on BTR1. As predicted in Section C.1.1, we observe that the density metric is consistently greater than the conservation metric in all experiments. Training with any variant of the DEEPMINIMIZER ($\mathcal{L}_{DM}$) loss generally lowers density for minimizer ($\mathcal{M}$) and complement ($\mathcal{C}_{w/2}$) schemes, but also decreases their conservation. On the contrary, the conservation of syncmers ($\mathcal{O}_{w/2}$) increases with training at the expense of raising its density. This is most likely because the sampling behavior of syncmers is not compatible with the above variants of $\mathcal{L}_{DM}$, which was originally designed for minimizers; whereas the complement mask behaves almost identically to the minimizer mask due to its mild sub-sampling.

The effectiveness of training masked minimizers.

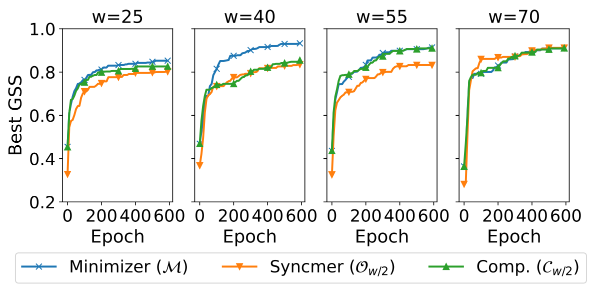
This experiment demonstrates that our proposed loss function $\mathcal{L}_{gss}$ learns robustly and improves GSS in various settings of w, k and different masks ν. Fig. C.2 plots the GSS of the masked minimizers $\mathcal{M}$, $\mathcal{O}_{w/2}$ and $\mathcal{C}_{w/2}$ over 600 training epochs in two settings: (1) $w = 15$ and $k \in [25, 40, 55, 70]$; (2) $k = 15$ and $w \in [25, 40, 55, 70]$. This experiment is repeated on two sequences, CHRXC and CHR1. All experiments show that GSS steadily increases over 600 training epochs by 1.5 to 5 times that of their initial random weights. We observe that the performance of minimizers ($\mathcal{M}$) is highly similar to the complement scheme ($\mathcal{C}_{w/2}$), except for $(w, k) = (15, 40)$ with CHR1 and $(w, k) = (15, 25), (40, 15)$ with CHRXC. Both these masks outperform syncmers ($\mathcal{O}_{w/2}$) in most settings, which corroborates the observation in the previous experiment. Appendix C.6 shows the individual effects of training on the conservation and density metrics



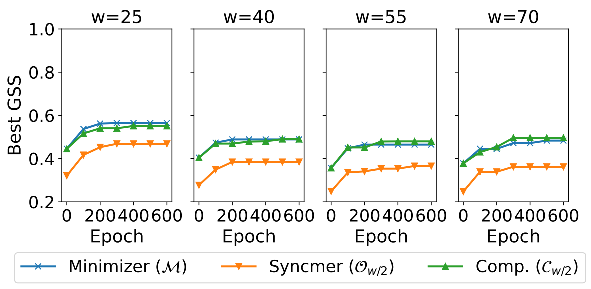
(a) CHRXC, $w = 15$



(b) CHR1, $w = 15$



(c) CHRXC, $k = 15$



(d) CHR1, $k = 15$

Figure C.2: Comparing GSS of different masked minimizers vs. number of training epochs on CHRXC and CHR1.

for the experiments in Fig. C.2(a), thus confirming our analysis in Section C.1.2.

Comparing GSS of compatible schemes with different training losses and masks.

This experiment compares the performance of various masked minimizers whose orderings are randomized; trained with 3 different DEEPMINIMIZER-based losses (i.e., $\mathcal{L}_{DM}$, $\mathcal{L}_{con}$ and $\mathcal{L}_{gss}$); or constructed using other optimization techniques such as MINICEPTION [85] and PASHA [18]. We compute the GSS on all combinations of $w \in \{10, 15, 20\}$ and $k \in \{10, 15\}$. Table C.1 summarizes the result of this study on CHRXC. Overall, the masked minimizer loss function $\mathcal{L}_{gss}$ performs the best, having achieved the highest GSS in 4 over 6 settings of w, k. Across 18 experiments (i.e., crossing 6 settings of (w, k) and 3 loss functions), the best GSS is achieved by the minimizer mask on 6 experiments, the syncmer mask on 1 experiment, and the complement mask on 4 experiments. In 10 out of these 11 experiments, with $(15, 15, \mathcal{L}_{gss})$ being the exception, the optimized mask ν_* achieves the same best GSS, which is reason-

	Conservation Loss $\mathcal{L}_{con}$				DeepMinimizer Loss $\mathcal{L}_{DM}$				Masked Minimizer Loss $\mathcal{L}_{gss}$			
(w,k)	$\mathcal{M}$	$\mathcal{O}_{w/2}$	$\mathcal{C}_{w/2}$	$\mathcal{V}$	$\mathcal{M}$	$\mathcal{O}_{w/2}$	$\mathcal{C}_{w/2}$	$\mathcal{V}$	$\mathcal{M}$	$\mathcal{O}_{w/2}$	$\mathcal{C}_{w/2}$	$\mathcal{V}$
$10,10$	**0.701**	0.656	0.685	**0.701**	0.716	0.529	0.722	**0.745**	**0.754**	0.601	0.753	**0.754**
$10,15$	**0.706**	0.650	0.696	**0.706**	0.768	0.656	**0.779**	**0.779**	0.719	0.566	0.704	**0.810**
$15,10$	0.693	0.621	0.680	**0.713**	0.690	0.613	0.726	**0.765**	0.683	0.597	0.701	**0.750**
$15,15$	0.715	0.667	0.710	**0.892**	0.813	0.746	**0.828**	**0.828**	0.812	**0.823**	0.817	0.817
$20,10$	**0.682**	0.614	0.679	**0.682**	0.677	0.608	0.673	**0.690**	**0.721**	0.595	0.690	**0.721**
$20,15$	0.747	0.716	**0.816**	**0.816**	0.829	0.714	**0.845**	**0.845**	**0.898**	0.794	0.894	**0.898**

Table C.1: Comparing GSS masked minimizers with 3 different training losses and 6 settings of (w,k) on CHRXC. The best GSS observed for each combination of (w,k) and loss function is given in **bold**.

able because our greedy mask pruning algorithm does not guarantee finding the optimal mask. In the remaining 7 experiments, ν_* outperforms all handcrafted masks, thus confirming the need to conduct this optimization step.

We repeat the same experiment for non-gradient optimization methods, including PASHA [18], MINICEPTION[85] and the random ordering baseline. We summarize their GSS performance in Table C.2. Among these methods, PASHA obtains the best GSS in 4 over 6 (w,k) settings, whereas MINICEPTION obtains the best GSS in the other 2 settings. While these methods outperform the random ordering baseline as expected, their performance is generally weaker than the gradient-based methods above. Similar to the previous experiment, we also observe that ν_* achieves the best on 17 over 18 settings with 10 being clear improvements over handcrafted masks. More interestingly, the syncmer mask obtains a GSS of 0.0 with MINICEPTION in many settings of (w,k), which suggests that none of the sampled locations is found at the $w/2$ offset. Although the cause of this is unclear, this phenomenon confirms the necessity of finding an optimal mask.

Exploiting the relative density metric.

This experiment demonstrates the exploitative behavior mentioned in Section C.1.1 via a special loss function $\mathcal{L}_{exploit} \triangleq \sum_{i=1}^{n} \Delta(\mathbf{f}(S_i;\alpha), \mathbf{f}(S;\alpha))$, which differs from $\mathcal{L}_{con}$ by swapping the template $\mathbf{g}(S;\beta)$

	MINICEPTION UHS				PASHA UHS				Random Ordering			
(w,k)	$\mathcal{M}$	$\mathcal{O}_{w/2}$	$\mathcal{C}_{w/2}$	$\mathcal{V}$	$\mathcal{M}$	$\mathcal{O}_{w/2}$	$\mathcal{C}_{w/2}$	$\mathcal{V}$	$\mathcal{M}$	$\mathcal{O}_{w/2}$	$\mathcal{C}_{w/2}$	$\mathcal{V}$
$10,10$	0.574	0.000	0.551	**0.582**	**0.629**	0.438	0.601	**0.629**	0.265	0.210	0.287	**0.288**
$10,15$	0.473	0.251	0.487	**0.501**	0.756	0.192	**0.769**	**0.769**	0.222	0.079	0.262	**0.265**
$15,10$	**0.557**	0.000	0.576	**0.557**	0.510	0.444	**0.589**	**0.589**	0.271	**0.276**	0.253	0.271
$15,15$	0.435	0.369	0.471	**0.509**	0.521	0.302	0.558	**0.632**	**0.172**	0.119	0.140	**0.172**
$20,10$	**0.604**	0.000	0.489	**0.604**	0.435	0.307	0.551	**0.553**	0.189	0.202	0.239	**0.247**
$20,15$	0.392	0.000	0.435	**0.476**	0.329	0.315	0.390	**0.399**	**0.132**	0.095	0.128	**0.132**

Table C.2: Comparing GSS of different masked minimizers with 3 different discrete construction methods and 6 settings of (w, k) on CHRXC. The best GSS observed for each combination of (w, k) and construction method is given in **bold**.

in each pairwise Δ-distance term with $\mathbf{f}(S;\alpha)$. The purpose of this substitution is to isolate any training signal for density (which is implicitly encoded in the template) and directly prioritize minimizing relative conservation. As minimizers schemes must select one position per (w, k)-window by construction, they do not suffer from this exploit. We thus train only the syncmer mask $\mathcal{O}_{w/2}$ on a random sequence with $L = 1000$, using $\mathcal{L}_{exploit}$ with $w = 10$ and $k = 15$.

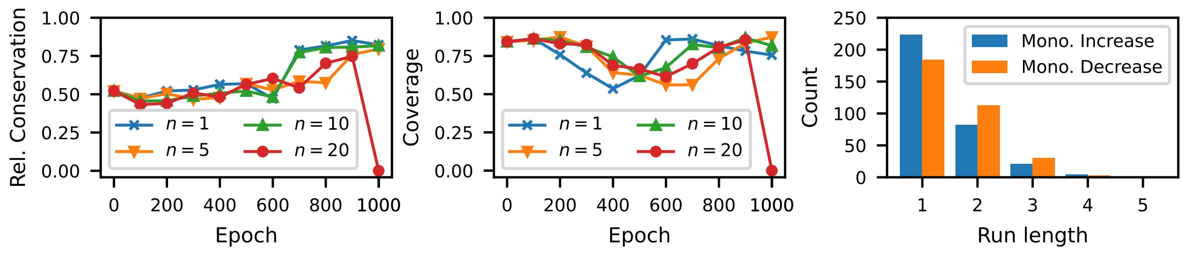


Figure C.3: Left: relative conservation and coverage of $\mathcal{O}_{w/2}$ trained on a random sequence using $\mathcal{L}_{exploit}$ with varying no. sampled mutations n; Right: no. segments with monotonically changing priority scores at each segment length.

Fig. C.3 (left) plots the relative conservation and coverage metrics obtained over 1000 epochs with different number of sampled mutations $n \in \{1, 5, 10, 20\}$ per training epoch. We observe that $\mathcal{L}_{exploit}$ consistently improves relative conservation as expected. When $n = 20$, the optimizer finds the exploit mentioned in Section C.1 after 1000 epochs, which causes both metrics to reach 0. The resulting sketch selects no k-mers (i.e., 0 coverage) and is trivially conserved when mutations are introduced (i.e., infinity conservation, manually set to 0). Fig. C.3 (right) plots the number of segments with monotonically increasing/decreasing priority scores at each segment length. The longest segment with monotonically decreasing priority score is 4, which is smaller than the syncmer offset $t = w/2 = 5$. This implies that all lowest scoring k-mers are found within the first $t - 1$ positions of their respective windows and none are sub-sampled into the syncmer sketch. Appendix C.6 repeats this experiment for $t = 6, 7, 8, 9$ and yields the same pattern with their discovered exploits, thus confirming the existence of our theoretical scenario in Section C.1.1 and the necessity of the GSS metric.

The minimizer mask is a good default configuration.

Fig. C.4 (left) shows the scatter plot of all 2^w masked minimizers trained on BTR4 using $\mathcal{L}_{gss}$ with $w = 10$ and $k = 15$, grouped by the number of 1-entries in their masks. Similar experiments on BTR1, BTR2 and BTR3 are deferred to Appendix C.6. We observe that the average GSS increases with the number of 1-entries in all experiments, which implies that the minimizer mask is a good default choice. However, we also observe that there exist specific masks that perform better than the minimizer mask, thus justifying mask optimization in certain applications.

Preventing repeated sampling in homopolymer-rich sequences.

One advantage of syncmers with $t > 1$ is the ability to avoid repeated sampling of identical k-mers in homopolymer substrings (i.e., substrings with repeated submer patterns) [16]. To confirm this, Fig. C.4 (right) plots the GSS of all syncmer masks and their complements on a synthetic sequence with $L = 100000$ and 0.2% homopolymer content. The dotted line shows the GSS of the minimizer mask, which expectedly performs worse than most syncmers (except for $t = 1$) due to the repeated sampling pitfall. Because of the left-most tie breaking rule, every masked minimizers with a 1 at the left most position of the mask (which includes minimizers, syncmers with t=1 and complement with $t \neq 1$) suffer from the pitfall of having a high density. This is indeed reflected in their similar GSS to the minimizer mask. On the contrary, we observe that the complement mask $\mathcal{C}_1$ (i.e., 1st position is pruned) achieves the best GSS of 0.56. This is because it avoids the repeated sampling pitfall in the same way any syncmer mask with $t > 1$ does, but otherwise performs like a minimizer scheme and does not suffer from the low coverage of syncmers.

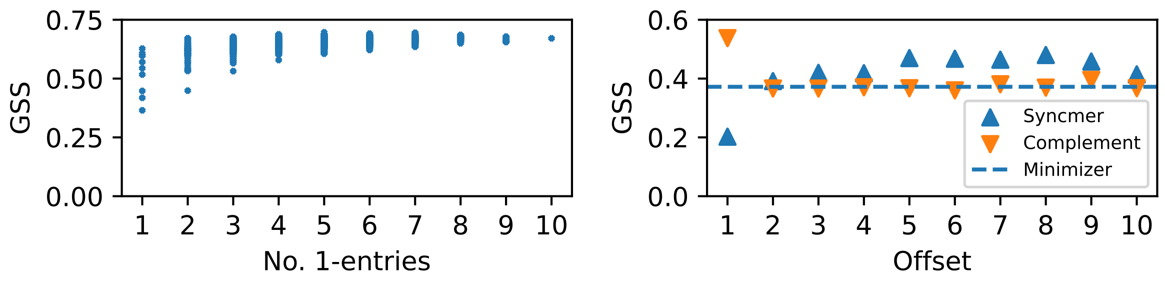


Figure C.4: Left: GSS vs. number of 1-entries of all mask minimizers trained on the bacterial genome BTR4; Right: GSS vs. offset positions of syncmer and complement masks on a synthetic sequence with high homopolymer content.

C.4 Details of benchmark sequences

Label	Description (Assembly)	Length
CHRXC	Centromere region of human chromosome X [62]	3106132
CHR1	Human chromosome 1	233587144
BTR1	Blautia producta (GCA_004210255.1)	6354838
BTR2	Blautia hansenii DSM 20583 (GCF_002222595.2)	3065949
BTR3	[Clostridium] scindens (GCA_009684695.1)	3785527
BTR4	Blautia producta ATCC 27340 = DSM 2950 (GCA_010669205.1)	6197116

Table C.3: Descriptions and lengths of sequences used in Section C.3.

C.5 Other implementation details

We implement our method using PyTorch and deploy all experiments on a RTX-3080 GPU. Due to limited GPU memory, each training epoch only computes the loss on a randomly sampled batch of 32 substrings of length $\ell = 1500$ bases. The conservation component of $\mathcal{L}_{gss}$ is averaged over 5 random mutations, simulated using a 10% base substitution rate. Evaluation of conservation is likewise obtained using 5 random mutations. Network weights are optimized using the ADAM optimizer [46] with default parameters.

C.6 Other results

Effectiveness of training on conservation and density metrics.

Fig. C.5 demonstrates the individual effect of training the proposed loss $\mathcal{L}_{gss}$ on the conservation and density metrics. We observe that both conservation and density of the syncmer mask are upper-bounded by that of the minimizer mask, which confirms the result of Theorem C.1.1. We observe that $\mathcal{L}_{gss}$ improves conservation but worsens density for the syncmer mask, which is similar to our first experiment. However, this is not the case for the minimizer and complement mask, which obtain significant improvements in both metrics over 600 training epochs. We note that this does not contradict our analysis, as conservation is still bounded by density at any point during the training. Rather, this implies that our method has found a favorable trade-off between the two metrics, which in turn explains the sharper increases in GSS compared to that of syncmer across all experiments.

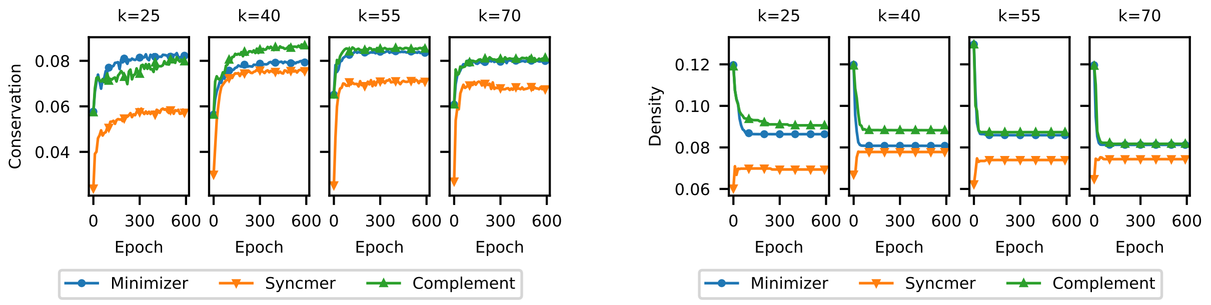


Figure C.5: Comparing conservation and density metrics of different masked minimizers vs. number of training epochs on ChrXC with $w = 15$ and $k \in \{25, 40, 55, 70\}$.

GSS profiles of minimizer masks on other bacterial genomes.

Fig. C.6 shows the scatter plots of all 2^w masked minimizers trained on BTR1, BTR2 and BTR3 using $\mathcal{L}_{gss}$ with $w = 10$ and $k = 15$, grouped by the number of 1-entries in their masks. We observe the same increasing pattern of average GSS with respect to number of 1-entries in the mask, thus confirming that the minimizer mask is indeed a good default choice.

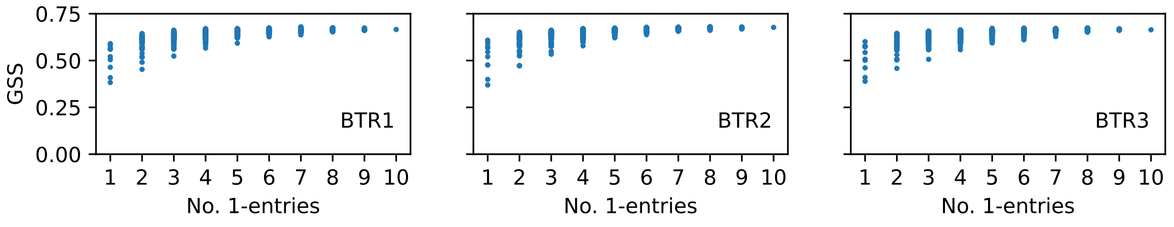


Figure C.6: GSS vs. number of 1-entries of all mask minimizers on bacterial genomes BTR1-3.

Exploiting the relative conservation metric with varying offsets.

We repeat this experiment for different syncmer masks with $t \in \{6, 7, 8, 9\}$ and plot all results in Fig. C.7. In all of these experiments, we observe that the model trained with $n = 20$ sampled mutations per epoch always find the exploit within $1000-1500$ epochs. The bar plots once again confirm that for each value of t, the exploitative solution contains no segment with more than $t - 1$ consecutively decreasing scores. We note that the total count for $t = 7$ is significantly lower than other values of t because the solution contains several segments of monotonically increasing scores that are relatively long, which count towards the > 6 bucket.

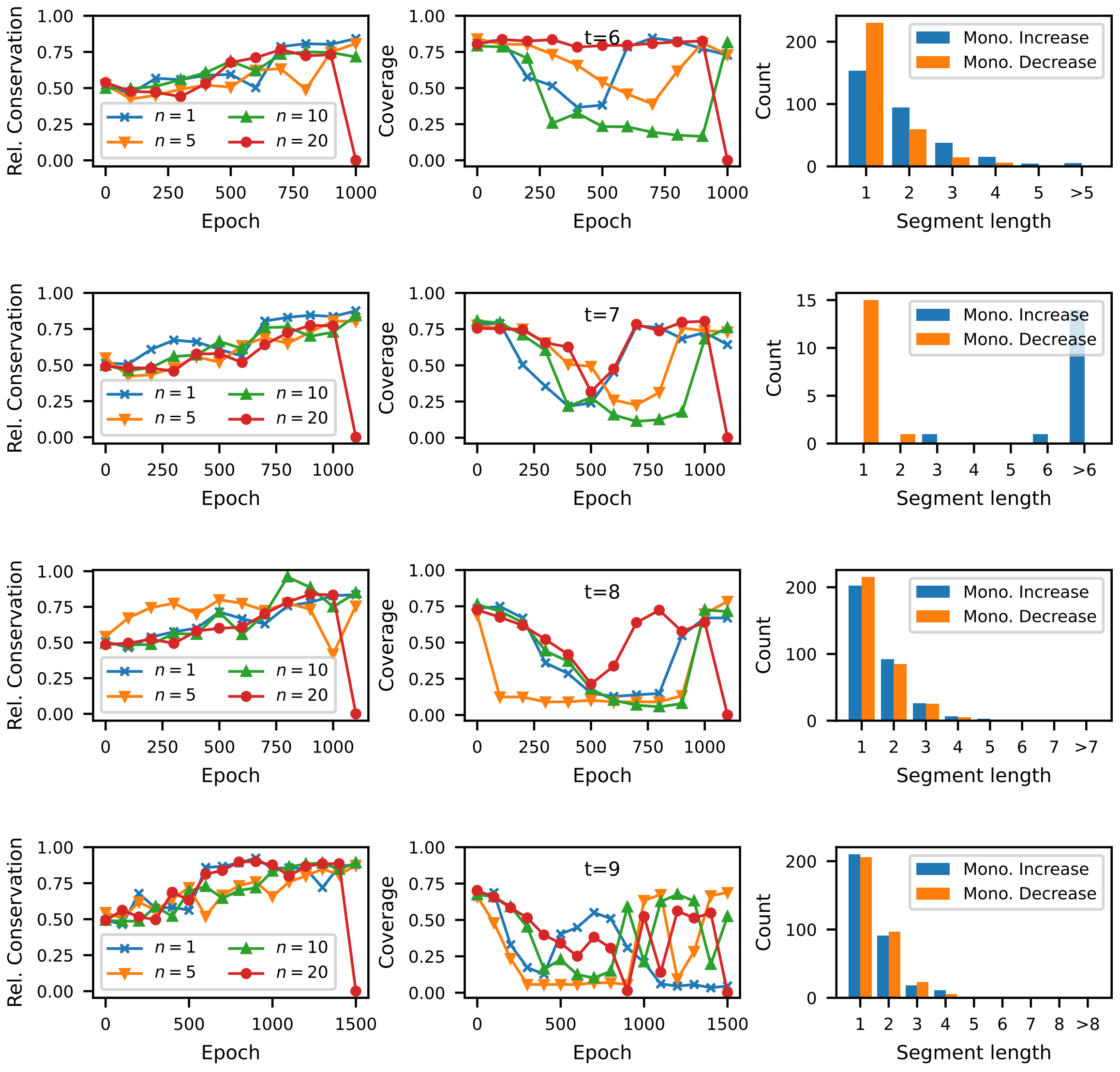


Figure C.7: Finding the relative conservation exploit for various syncmer masks with (from top to bottom) offset $t \in \{6, 7, 8, 9\}$ with $\mathcal{L}_{exploit}$, $w = 10$ and $k = 15$.

Appendix D

Neural Architecture Search Appendix

D.1 Personalized Federated Neural Architecture Search

Our method, FEDPNAS, adopts the same over-parameterized architecture space as suggested by Hu et al. [35]. Unlike [35], which lacks the ability to customize architectures for individual tasks, our method achieves architecture personalization via factorizing the architecture space into: (1) a base component that is shared among all client models; and (2) a task-specific component, which personalizes across clients to best solve their respective tasks (Section D.1.1). We learn the operator mixing weights $\bar{m}(e)$ for all edges, which induce the joint distribution over all edge operators over two phases.

In the *federated* phase of FEDPNAS, a common overall architecture is learned such that all task-specific architectures can be quickly and optimally derived from it with minimal fine-tuning efforts. This learning objective is achieved by extending the MAML meta learning algorithm proposed by Finn et al. [21]. (Section D.1.2). Subsequently, in the *adaptation* phase of FEDPNAS, each client separately performs local update of its task-specific component using private data. Unlike other meta learning objectives [21], which only focus on adapting the weights of a single architecture, this adaptation step will result in diverging personalized architectures that are optimal for their respective tasks. The remainder of this section describes these contributions in details.

D.1.1 Cell-based Architecture Space

Similar to Hu et al. [35], we express the over-parameterized architecture $\mathcal{G}_{\mathcal{O}}$ as a stack of modular cells, i.e., $\mathcal{G}_0 = \bigcup_{t=1}^{C} \mathcal{G}_t$, where C is the total number of cells and each cell $\mathcal{G}_t$ denotes a compact space of architecture constructed from the operator list $\mathcal{O}$. To obtain an architecture from $\mathcal{G}_{\mathcal{O}}$, we first distill an architecture module $G_t = (V_t, E_t)$ from each cell by selecting a subset of edges in $\mathcal{G}_t$. The propagation path of the input vector across these modules is then heuristically chosen and specified by a cell-level DAG. For example, a simple linear propagation scheme can be written as $G(\mathbf{x}) = G_C \circ G_{C-1} \cdots \circ G_1(\mathbf{x})$, where $\mathbf{x}$ denotes an input vector and each G_i is treated as a feature map.

The inner computation of each distilled module G_t is similarly defined as in Section 4. Given an input vector $\mathbf{x}$, we then sample an operator o_e on each edge $e \equiv (v, v') \in E_t$ from the vocabulary $\mathcal{O} \triangleq [o_1, o_2 \ldots o_D]$ using a parameterized categorical distribution p_e^t. Hu et al. [35] assumes this edge sampling distribution is independent of the input $\mathbf{x}$ and any intermediate feature mapping of $\mathbf{x}$, hence does not make use of these context information in deciding the distilled architecture. In the context of *horizontal* NAS, we instead argue that these information are valuable in guiding client-level architecture adaptations and consequently parameterize p_e^t as conditional sampling distributions, which jointly model a hierarchical decision process (Section D.1.3).

The feature aggregation scheme for any arbitrary node $v' \in V_t$ is then given as:

$$z_{v'} = \sum_{e \equiv (v,v')}^{e \in E_t} \mathbb{E}_{r \sim p_e^t}\left[r^\top \xi(z_v)\right] \simeq \sum_{e \equiv (v,v')}^{e \in E_t} \sum_{i=1}^{s} r_{ei}^\top \, \xi(z_v) \,, \tag{D.1}$$

where each r_{ei} is a one-hot vector drawn from p_e^t and $\xi(z_v) \triangleq [o_1(z_v), o_2(z_v) \ldots o_D(z_v)]$ is the concatenation of all possible transformations of z_v using the operators in $\mathcal{O}$. Even though the computation above requires sampling from a categorical distribution, it can be made differentiable with respect to the parameters of p_e^t using the straight through Gumbel-softmax trick [42].

Unlike the original design, which assumes similar importance for every cell in the architecture stack, we opt to split our cell-based architecture into two component stacks with different roles to facilitate our personalized architecture search goal. Specifically, our search space contains: (a) a *base stack* $\mathcal{G}_b = \{\mathcal{G}_b^1, \mathcal{G}_b^2 \ldots \mathcal{G}_b^{C_b}\}$, which aims to capture a feature map that is universally useful to all tasks; and (b) a *personalized stack* $\mathcal{G}_p = \{\mathcal{G}_p^1, \mathcal{G}_p^2 \ldots \mathcal{G}_p^{C_p}\}$, which will be adapted using client data to capture task-specific features. Fig. D.1 summarizes the relationship of these components.

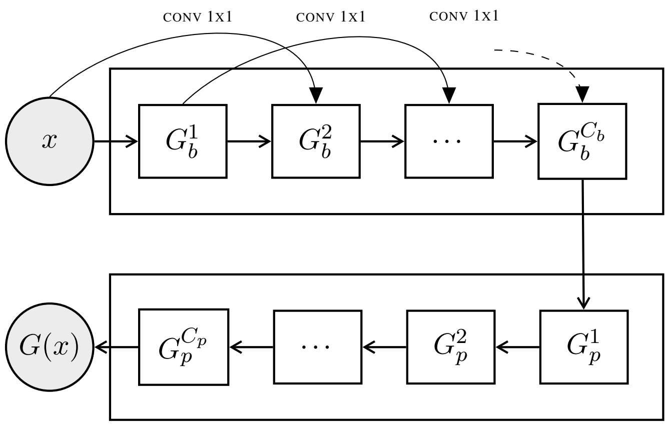


Figure D.1: Feature mapping induced by the component stacks of our architecture space. Each cell in the base stack receives outputs from two previous cells, whereas each cell in the personalized stack receives outputs from only one previous cell.

Generally, we assume that the tasks are broadly related and diverge in finer details. Therefore, the base architecture stack is designed to be significantly more expressive than the personalized stack, via employing a larger operator vocabulary and a more sophisticated propagation scheme as shown in Fig. 4.1. In addition, as we will subsequently discuss in Section D.1.2, our objective is formulated such that its gradient evaluation will require approximating the Hessian of the personalized component. As such, a minimally expressive personalized stack is also a practical design to lower the computational cost.

D.1.2 Personalized Federated Learning Objective

In this section, we will now discuss the learning objective of FEDPNAS, which enables the discovery of personalized architecture. Let $\theta = \{\theta_b, \theta_p\}$ respectively denote all trainable parameters of the two component stacks above. In particular, $\theta_b = \{W_b, \Pi_b\}$ contains the concatenated weights W_b of all edge operators; and the concatenated parameters $\Pi_b = [\Pi_e^t]_{t\in[C_b],e\in E_t}$ of the joint edge sampling distribution. Likewise, $\theta_p = \{W_p, \Pi_p\}$ contains their counterparts in the personalized stack. Given N local clients with tasks $\{\Omega_1, \Omega_2 \ldots \Omega_N\}$, the straight-forward extension of FL objective to the NAS problem defined by this search space can be written as:

$$\theta_* = \underset{\theta}{\text{argmax}} \frac{1}{N} \sum_{i=1}^{N} F_{\Omega_i}(\theta) \tag{D.2}$$

McMahan et al. [61] proposes a privacy preserving approach to optimize this objective by alternating between two communication steps of the model parameters. At any iteration $t \geq 0$:

- Each local client performs a local gradient step with its current parameter θ_i^t and sends the suggested update $\bar{\theta}_i^t = \theta_i^t + \lambda \nabla_\theta F_{\Omega_i}(\theta_i^t)$ to a central server.
- The central server then computes $\theta_{\text{SERVER}}^t = \frac{1}{N}\sum_{i=1}^N \bar{\theta}_i^t$ and broadcasts the aggregated parameters θ_{SERVER}^t to all clients.
- Each local client performs the update $\theta_i^{t+1} \leftarrow \theta_{\text{SERVER}}^t$ and prepares for the $(t+1)^{\text{th}}$ communication round.

This FL scheme implies that all clients will follow the same architecture distribution after the last communication round, which is not necessarily optimal in a heterogeneous task setting. To address this, our framework instead adopts the MAML meta learning objective [21], which aims to find a favorable initialization of θ that yields maximum averaged performance *given an expected adaptation step*. This is different from Eq. D.2, which does not directly optimize for this initialization, but approximates it using the instantaneous averaged performance. Explicitly, this is achieved by the following objective:

$$\theta_* = \underset{\theta_b, \theta_p}{\arg\max} \frac{1}{N} \sum_{i=1}^{N} F_{\Omega_i}\left(\theta_b, \theta_p + \lambda \nabla_{\theta_p} F_{\Omega_i}(\theta_b, \theta_p)\right) , \tag{D.3}$$

in which the difference from Eq. (D.2), highlighted in red, models a gradient ascent update with step size λ to the aggregated personalized parameters θ_p, which is to be conducted by each client at the end of the federated phase. Intuitively, this gradient step acts as a regularization term which favors θ that are simultaneously close to all task-specific optima and yield the best averaged performance after the adaptation phase.

We now derive the gradient of this objective and discuss a practical algorithm to perform its computation. First, we let $\tilde{\theta}^t \triangleq \left(\theta_b^t, \theta_p^t + \lambda \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t)\right)$ denote the anticipated update of θ at time t of some arbitrary client. Then, dropping the client index for clarity, we subsequently derive the gradient ascent update for each client pertaining to the above personalized FL objective:

$$\begin{aligned}
\bar{\theta}^t &= \theta^t + \lambda \nabla_{\theta^t} F_\Omega\left(\tilde{\theta}^t\right) \\
&= \theta^t + \left(\lambda \nabla_{\theta^t} \tilde{\theta}^t\right)\left(\nabla_{\tilde{\theta}^t} F_\Omega(\tilde{\theta}_t)\right) \\
&= \theta^t + \begin{bmatrix} \lambda \mathbf{I} & \lambda^2 \nabla_{\theta_b^t} \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \\ \mathbf{0} & \lambda^2 \nabla^2_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \end{bmatrix} \left(\nabla_{\tilde{\theta}^t} F_\Omega(\tilde{\theta}^t)\right) \ ,
\end{aligned} \tag{D.4}$$

where we applied chain rule in the second equality and subsequently expanded $\lambda \nabla_{\theta^t} \tilde{\theta}^t$ in the third equality. The second-order gradient terms $\nabla_{\theta_b^t} \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t)$ and $\nabla^2_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t)$, however, are expensive to evaluate exactly. In order to derive practical computations of these terms, we note that the first-order Taylor approximation of the Hessian $\nabla^2_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t)$ can be written as:

$$\begin{aligned}
\nabla^2_{\theta^t} F_\Omega(\theta_b^t, \theta_p^t) &= \begin{bmatrix} \nabla^2_{\theta_b^t} F_\Omega(\theta_b^t, \theta_p^t) & \nabla_{\theta_b^t} \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \\ \nabla_{\theta_p^t} \nabla_{\theta_b^t} F_\Omega(\theta_b^t, \theta_p^t) & \nabla^2_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \end{bmatrix} \\
&\simeq \begin{bmatrix} \nabla_{\theta_b^t} F_\Omega(\theta_b^t, \theta_p^t) \\ \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \end{bmatrix} \begin{bmatrix} \nabla_{\theta_b^t} F_\Omega(\theta_b^t, \theta_p^t) \\ \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \end{bmatrix}^\top \\
&= \begin{bmatrix} \nabla_{\theta_b^t} F_\Omega(\theta_b^t, \theta_p^t) \nabla^\top_{\theta_b^t} F_\Omega(\theta_b^t, \theta_p^t) & \nabla_{\theta_b^t} F_\Omega(\theta_b^t, \theta_p^t) \nabla^\top_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \\ \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \nabla^\top_{\theta_b^t} F_\Omega(\theta_b^t, \theta_p^t) & \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \nabla^\top_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \end{bmatrix} \ .
\end{aligned} \tag{D.5}$$

Matching appropriate terms in the above derivation then implies the following approximations, which can be computed with a single forward-backward pass of the architecture:

$$\nabla_{\theta_b^t} \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \simeq \nabla_{\theta_b^t} F_\Omega(\theta_b^t, \theta_p^t) \nabla^\top_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \ , \tag{D.6}$$

$$\nabla^2_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \simeq \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \nabla^\top_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \ . \tag{D.7}$$

Plugging this back to Eq. (D.4) gives:

$$\bar{\theta}^t \simeq \theta^t + \begin{bmatrix} \lambda \mathbf{I} & \lambda^2 \nabla_{\theta_b^t} F_\Omega(\theta_b^t, \theta_p^t) \nabla^\top_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \\ \mathbf{0} & \lambda^2 \nabla_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \nabla^\top_{\theta_p^t} F_\Omega(\theta_b^t, \theta_p^t) \end{bmatrix} \left(\nabla_{\tilde{\theta}^t} F_\Omega(\tilde{\theta}^t)\right) \ , \tag{D.8}$$

where the term $\nabla_{\tilde{\theta}^t} F_\Omega(\tilde{\theta}^t)$ requires another forward-backward pass to compute (i.e., $\tilde{\theta}^t$ is computed in the same pass with the two approximated gradient terms above). Overall, this results in a local update scheme which uses two forward-backward passes of the architecture per iteration:

- Compute $F_\Omega(\theta^t)$ in the first forward pass.
- Perform backpropagation to obtain $\nabla_{\theta^t} F_\Omega(\theta^t) = \left[\nabla_{\theta^t_b} F_\Omega(\theta^t_b, \theta^t_p), \nabla_{\theta^t_p} F_\Omega(\theta^t_b, \theta^t_p)\right]$.
- Compute $\nabla_{\theta^t_b} F_\Omega(\theta^t_b, \theta^t_p)\nabla^\top_{\theta^t_p} F_\Omega(\theta^t_b, \theta^t_p)$ and $\nabla_{\theta^t_p} F_\Omega(\theta^t_b, \theta^t_p)\nabla^\top_{\theta^t_p} F_\Omega(\theta^t_b, \theta^t_p)$.
- Compute $\tilde{\theta}^t = \left(\theta^t_b, \theta^t_p + \lambda \nabla^\top_{\theta^t_p} F_\Omega(\theta^t_b, \theta^t_p)\right)$.
- Compute $F_\Omega(\tilde{\theta}^t)$ in the second forward pass.
- Perform backpropagation to obtain $\nabla_{\tilde{\theta}^t} F_\Omega(\tilde{\theta}^t)$.
- Compute the update in Eq. (D.8).

D.1.3 Context-Aware Operator Sampling

Finally, this section describes the parameterization of the joint operator sampling distribution over the edges of $\mathcal{G}_O$. Let $O_G \triangleq \cup_{t\in[C]} O_{G_t}$ be the set of all selected edge operators in a distilled architecture G, where $O_{G_t} \triangleq \{o^t_e \in \mathcal{O}\}_{e\in E_t}$ in turn denotes the set of all selected edge operators in cell G_t (i.e., in a single stack architecture). Hu et al. [35] then assumes a fully factorizable sampling distribution:

$$p(O_G) \;=\; \prod_{t\in[C]} p(O_{G_t}) \;=\; \prod_{t\in[C]}\prod_{e\in E_t} p^t_e(o^t_e; \Pi^t_e)\;, \tag{D.9}$$

where Π^t_e denotes the learnable parameters of the edge sampling distribution p^t_e for every $t \in [C]$ and $e \in E_t$. This formulation, however, does not factor in the important context information carried by the input instance $\mathbf{x}$ and its subsequent embeddings as $\mathbf{x}$ propagates through G. While this approach might be sufficient when only one architecture needs to be distilled from $\mathcal{G}_\mathcal{O}$, it is challenging to extend to the *horizontal* NAS setting because every edge in the over-parameterized architecture $\mathcal{G}_\mathcal{O}$ is required to maintain and optimize its own set of categorical distribution parameters. In the context of our *horizontal* NAS problem, this means that the set of parameters that need to be personalized will also scale with the size of the architecture, hence posing both a computational and a convergence problem.

To overcome this issue, we will instead look at the conditional sampling distribution $p(O_g \mid \mathbf{x})$ and subsequently model architecture distillation as a Markov chain decision process, where each cell in the architecture stack will sequentially determine its edge operators given the input feature it receives from previous cells. For simplicity, we give our factorization of $p(O_G \mid \mathbf{x})$ below as if the architecture contains a single stack of cells and uses a linear propagation scheme, but it would be trivial to extend this formulation to any other propagation scheme:

$$\begin{aligned} p(O_G \mid \mathbf{x}) &= p(O_{G_1} \mid \mathbf{x}) \prod_{t=2}^{C} p(O_{G_t} \mid O_{G_{t-1}}, \mathbf{x}) \\ &\simeq \prod_{t=1}^{C} \prod_{e \equiv (v,v')}^{e \in E_t} p_e^t \left(o_{v,v'}^t \mid z_v \right) \end{aligned} \tag{D.10}$$

where z_v denotes the feature vector at an arbitrary node v in G and in the factorization above, we have also assumed that z_v acts as the sufficient statistics of the random variable $o_{v,v'}^t$.

The advantage of this formulation is two-fold. First, it provides a natural mechanism to incorporate context information into the edge selection process, thus allowing FEDPNAS to efficiently memorize architecture distillation patterns across clients. Second, the above factorization of $p(O_G \mid \mathbf{x})$ as a product of edge-wise conditional probabilities further reveals a compact representation of all sampling parameters using a single neural network ψ, which reduces the number of parameters that require fine-tuning in the local adaptation phase. That is, for any edge $e \equiv (v, v') \in E_t$, its context-aware operator distribution conditioned on z_v is given as $p_e^t(o_{v,v'}^t \mid z_v) \triangleq \text{Cat}\left(D, \psi\left(z_v\right)\right)$, where D denotes the number of operators in $\mathcal{O}$ and the neural network ψ maps the z_v to the event probabilities of a categorical distribution.

D.2 Experiments

This section describes our experiments to showcase the performance of FEDPNAS compared to other NAS and FL benchmarks. Our empirical studies are conducted on two image recognition datasets: (a) the CIFAR-10 dataset [47] which aims to predict image labels from 10 classes given a train/test set of $50000/10000$ colour images of dimension 32×32 pixels; and (b) the MNIST dataset [49] which aims to predict handwritten digits (i.e. 0 to 9) given a train/test set of $60000/10000$ grayscale images of dimension 28×28 pixels. Our search space entails 2^{40} possible architectures, which is detailed in Appendix D. We compare two variants of our framework, CA-FEDPNAS (with context-aware operation sampler) and FEDPNAS (without the operation sampler), against: (a) FEDAVERAGING of a fixed architecture to justify the need for NAS in FL; (b) FEDDSNAS - which trivially extends DSNAS to the FL setting (Eq. (D.2)); and finally (c) CA-FEDDSNAS, which extends FEDDSNAS with our context-aware sampler.

D.2.1 Heterogeneous predictive tasks

We first design a control experiment to test our framework on heterogeneous tasks and demonstrate the necessity of architecture personalization. To simulate this scenario, we first distribute the data i.i.d across clients (10000/2000 and 12000/2000 training/test images per client for CIFAR-10 and MNIST datasets respectively). Then, we independently apply a different transformation to each partitioned dataset. Input images within the same train/test set is subject to the same transformation. In both our experiments, the client datasets are subjected to rotations of $-30°, -15°, 0°, 15°$ and $30°$ respectively. Fig. D.2 below shows the performance of all the methods in comparison, plotted against number of search epochs and averaged over the above rotated variants of CIFAR-10 and MNIST datasets.

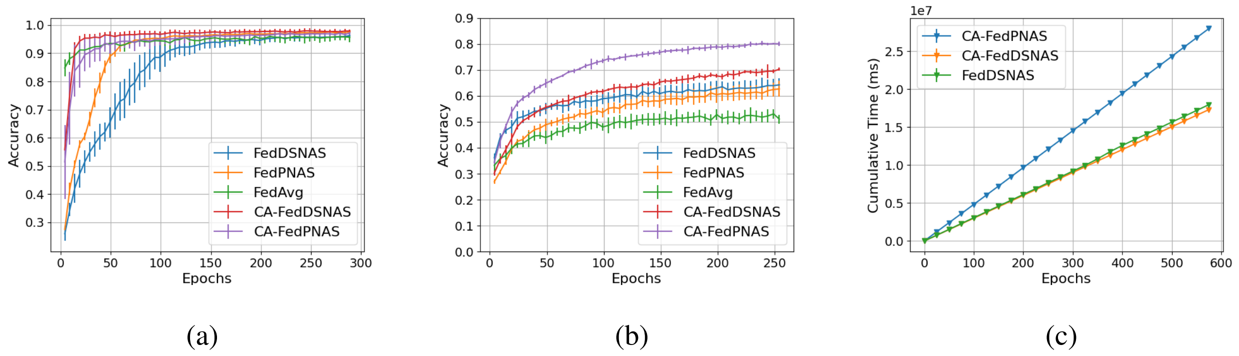


Figure D.2: Plotting average classification accuracy of various methods against no. training epochs on heterogeneous tasks derived from (a) MNIST and (b) CIFAR-10 dataset. Figure (c) compares cumulative running time of various methods against no. training epochs on the CIFAR-10 dataset.

On the MNIST dataset (Fig. D.2b), all methods converge to a similar performance. Among the NAS benchmarks, FEDPNAS and FEDDSNAS both converge slower than FEDAVG and start off with worse performance in early iterations. This is expected since FEDAVG does not have to search for the architecture and it is likely that the default architecture is sufficient for the MNIST task. On the other hand, we observe that both CA-FEDPNAS and CA-FEDDSNAS converge much faster than their counterparts without the context-aware operation sampler component. This shows that making use of contextual information helps to quickly locate regions of high-performing architectures, especially on similar inputs.

On the CIFAR-10 dataset (Fig. D.2a), we instead observe significant gaps between the worst per-

Heterogeneity	Task Description	FedDSNAS	CA-FedPNAS
Low	Rotate -30	0.947	**0.978**
	Rotate -15	0.973	**0.976**
	Vanilla	**0.988**	0.985
	Rotate 15	0.986	**0.987**
	Rotate 30	0.972	**0.981**
High	HueJitter -0.5	0.966	**0.978**
	HueJitter 0.5	0.967	**0.972**
	Vanilla	0.988	**0.989**
	Rotate -90	0.892	**0.932**
	Rotate 90	0.866	**0.932**

Table D.1: Predictive accuracy of CA-FedPNAS FedDSNAS on tasks with varying heterogeneity levels. Rotate X denotes a rotation transformation of X° on client data; Vanilla denotes the original MNIST images; and HueJitter X denotes a hue jitter transformation of training images by a factor of X. The best performance in each row is in bold font.

forming FedAvg and other NAS methods. This is likely because the default architecture does not have sufficient learning capability, which confirms the need for customizing solutions. Among the NAS benchmarks, we again observe that both CA-FedPNAS and CA-FedDSNAS outperform their counterparts without our operation sampler, which confirms the intuition above. Most remarkably, our proposed framework CA-FedPNAS achieves the best performance (0.8) and significantly outperformed both variants of federated DSNAS (0.71 for CA-FedDSNAS and 0.63 for FedDSNAS).

Lastly, Fig. D.2c shows the runtime comparison between three methods on the CIFAR-10 experiment. In terms of sampling time, we observe that there is negligible overhead incurred by using our context-aware sampler (CA-FedDSNAS vs. FedDSNAS). The time incurred by our update (CA-FedPNAS) scales by a constant factor compared to CA-FedDSNAS since we use exactly one extra forward-backward pass per update.

D.2.2 Tasks with varying heterogeneity levels

We expand the above study by subsequently investigating the respective performances of CA-FEDPNAS and FEDDSNAS on tasks with varying levels of heterogeneity. At low levels of heterogeneity, we deploy these methods on 5 sets of slightly rotated MNIST images. At high levels of heterogeneity, we employ a more diverse set of transformations on MNIST images, such as hue jitter and large angle rotations of 90° and -90°. Table D.1 shows the respective result of each task from these two settings. We observe that our method CA-FEDPNAS achieves better performance on most tasks and the performance gaps on tasks with higher heterogeneity are more pronounced (i.e., up to 7% improvement on ROTATE 90 task). This clearly shows the importance of architecture personalization when the training tasks are significantly different and justifies our research goal.

D.2.3 Knowledge transfer to completely new tasks

Finally, we conduct an ablation study to assess the quality of the *pre-adaptation* architecture distributions respectively discovered by CA-FEDPNAS and FEDDSNAS. In particular, we will leverage these learned distributions, which supposedly capture the broad commonalities of the task distribution, to generalize to completely unseen tasks (i.e., tasks that do not participate in the federated learning phase). To simulate this scenario, we train both methods on five clients whose local data consist of 12000 rotated CIFAR-10 images (i.e., in the range of $\pm 30^\circ$), similar to the setting of the first experiment. During the evaluation phase, however, we supply each local client with 2000 test images subjected to related but completely unseen transformations (i.e., 90° and -90° rotations).

UNSEEN TASK DESCRIPTION	FEDDSNAS	CA-FEDPNAS	FEDDSNAS (RETRAINED)	CA-FEDPNAS (RETRAINED)
ROTATE -90	0.545 ± 0.04	0.578 ± 0.09	0.699 ± 0.12	$\mathbf{0.734 \pm 0.17}$
ROTATE 90	0.553 ± 0.12	0.569 ± 0.06	0.673 ± 0.13	$\mathbf{0.727 \pm 0.22}$

Table D.2: Predictive accuracy (averaged over 5 clients) and standard deviation of CA-FEDPNAS and FEDDSNAS on two unseen tasks (CIFAR-10).

We summarize our results in Table D.2 above. First, we measure the performance of CA-FEDPNAS and FEDDSNAS without any weight retraining. When received no additional information from the unseen tasks, both methods perform poorly as expected. While CA-FEDPNAS achieves better predictive accuracy, the performance gap in this scenario is negligible. To provide additional clues for adaptation, albeit minimal, we retrain the weights of each local model with 200 images that are rotated according to their respective unseen task description. With only 100 retraining iterations on limited data, CA-FEDPSNAS already outperforms FEDDSNAS (5% and 8% improvement respectively on two unseen tasks). This implies that CA-FEDPNAS has captured more accurately the broad similarity of the task spectrum and requires minimal additional information to successfully adapt to unseen tasks.

Bibliography

[1] Maria-Florina Balcan, Vaishnavh Nagarajan, Ellen Vitercik, and Colin White. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. In *Conference on Learning Theory*, pages 213–274. PMLR, 2017. 1.2

[2] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International conference on machine learning*, pages 344–353. PMLR, 2018. 1.2

[3] 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. 1.1

[4] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., 1998. 1.2

[5] Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, Andrea Schaerf, and Tommaso Urli. Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers & Operations Research*, 65:83–92, 2016. 1.2

[6] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 550–559. PMLR, 2018. 1.2, 4.1, 4.2, 4.2.2

[7] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2), 2012. 1.2

[8] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011. 1.2

[9] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical science*, 8(1):10–15, 1993. 1.2

[10] Jiayi Chen and Aidong Zhang. Hetmaml: Task-heterogeneous model-agnostic meta-learning for few-shot learning across modalities. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 191–200, 2021. 5.3

[11] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016. 3.2.2, B.1.1

[12] Dan DeBlasio, Fiyinfoluwa Gbosibo, Carl Kingsford, et al. Practical universal k-mer sets for minimizer schemes. In *Proceedings of the 10th ACM Conference on Bioinformatics, Computational Biology*, pages 167–176. Association for Computing Machinery, 2019. 3.2.2, B.1.1

[13] Dan Deblasio, Kwanho Kim, and Carl Kingsford. More accurate transcript assembly via parameter advising. *Journal of Computational Biology*, 27(8):1181–1189, 2020. 1.2

[14] Sebastian Deorowicz, Marek Kokot, Szymon Grabowski, et al. KMC 2: fast and resource-frugal k-mer counting. *Bioinformatics*, 31(10):1569–1576, 2015. 3.1

[15] D. Duvenaud, J. Lloyd, R. Grosse, J. Tenenbaum, and G. Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 1166–1174, 2013. 1.1, 1.2, 1.3, 2.1, 2.2.1, 2.2.2, 2.3, A.1, A.2

[16] Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, 2021. 1.3, 3.1, 3.1, 3.1, 3.4.1, 5.2, C.1.2, C.3, C.3

[17] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics, 407-499*, 2004. 2.5, A.2, A.2.2

[18] Barış Ekim, Bonnie Berger, and Yaron Orenstein. A randomized parallel algorithm for efficiently finding near-optimal universal hitting sets. In *Proceedings of the* 24^{th} *Annual International Confer-*

ence on Research in Computational Molecular Biology, 2020. 1.3, 3.2.1, 3.3.1, 3.5, B.1.1, B.2.1, C.2, C.3, C.3, C.3

[19] Marius Erbert, Steffen Rechner, and Matthias Müller-Hannemann. Gerbil: a fast and memory-efficient k-mer counter with GPU-support. *Algorithms for Molecular Biology*, 12(1):1–12, 2017. 3.1

[20] A. Fallah, A. Mokhtari, and A. Ozdaglar. Personalized federated learning: Model-agnostic meta-learning approach. In *Proc. NeurIPS*, 2020. 4.3

[21] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proc. ICML*, pages 1126–1135, 2017. 1.3, 4.3, D.1, D.1.2

[22] David B Fogel. An introduction to simulated evolutionary optimization. *IEEE transactions on neural networks*, 5(1):3–14, 1994. 1.2

[23] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010. 1.2

[24] Tatsuo Fukagawa and William C. Earnshaw. The centromere: chromatin foundation for the kinetochore machinery. *Developmental Cell*, 30(5):496–508, 2014. B.2.5

[25] Y. Gal and R. Turner. Improving the Gaussian process sparse spectrum approximation by representing uncertainty in frequency inputs. In *Proc. ICML*, pages 655–664, 2015. 5.1

[26] Chaoyang He, Murali Annavaram, and Salman Avestimehr. FedNAS: Federated deep learning via neural architecture search. *arXiv e-prints*, pages arXiv–2004, 2020. 1.1, 1.2, 4.2.3, 4.5

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1.1

[28] J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for big data. In *Proc. UAI*, pages 282–290, 2013. (document), 5.1, A.2, A.3, A.2.2

[29] Minh Hoang, Nghia Hoang, Hai Pham, and David Woodruff. Revisiting the sample complexity of sparse spectrum approximation of gaussian processes. *Advances in Neural Information Processing*

Systems, 33:12710–12720, 2020. 5.1

[30] Minh Hoang, Guillaume Marcais, and Carl Kingsford. Masked minimizers: Unifying sequence sketching methods. *bioRxiv*, 2022. 5.2

[31] Minh Hoang, Hongyu Zheng, and Carl Kingsford. DeepMinimizer: A differentiable framework for optimizing sequence-specific minimizer schemes. In *International Conference on Research in Computational Molecular Biology*, pages 52–69. Springer, 2022. C.2, C.2, C.3

[32] Nghia Hoang, Thanh Lam, Bryan Kian Hsiang Low, and Patrick Jaillet. Learning task-agnostic embedding of multiple black-box experts for multi-task model fusion. In *International Conference on Machine Learning*, pages 4282–4292. PMLR, 2020. 4.3

[33] Trong Nghia Hoang, Kian Hsiang Low, Patrick Jaillet, and Mohan Kankanhalli. Active learning is planning: Nonmyopic ε-bayes-optimal active learning of gaussian processes. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 494–498. Springer, 2014. 5.1

[34] Martin Hofmann. Support vector machines-kernels and the kernel trick. *Notes*, 26(3):1–16, 2006. 2.1

[35] Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. DSNAS: Direct neural architecture search without parameter retraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12084–12092, 2020. 4.1, 4.2.3, 4.4, D.1, D.1.1, D.1.3

[36] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011. 1.2

[37] J. Idrissi, M. Amine, R. Hassan, G. Youssef, and E. Mohamed. Genetic algorithm for neural network architecture optimization. In *International Conference on Logistics Operations Management*, pages 1–4, 05 2016. doi: 10.1109/GOL.2016.7731699. 1.1

[38] Tomoharu Iwata and Atsutoshi Kumagai. Meta-learning from tasks with heterogeneous attribute spaces. *Advances in Neural Information Processing Systems*, 33:6053–6063, 2020. 5.3

[39] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018. 5.1

[40] Chirag Jain, Arang Rhie, Haowen Zhang, et al. Weighted minimizer sampling improves long read mapping. *Bioinformatics*, 36(Supplement_1):i111–i118, 2020. 3.2.2, B.1.1

[41] Chirag Jain, Arang Rhie, Nancy Hansen, et al. Long-read mapping to repetitive reference sequences using Winnowmap2. *Nature Methods*, 19:1–6, 2022. doi: 10.1038/s41592-022-01457-8. 3.1

[42] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 4.2.3, D.1.1

[43] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. Xing. Neural architecture search with Bayesian optimisation and optimal transport. In *Conference on Neural Information Processing Systems (NeuRIPS)*, 02 2018. 1.2

[44] C. D. Keeling and T. P. Whorf. Atmospheric carbon dioxide concentrations derived from flask air samples at sites in the SiO network. https://www.openml.org/d/41187, 2004. 2.5, A.2, A.2.2

[45] D. Kingma and M. Welling. Auto-Encoding Variational Bayes. In *Proc. ICLR*, 2013. 2.2.2, 2.3, 5.1

[46] Diederik P. Kingma and Jimmy Ba. ADAM: A Method for Stochastic Optimization. *Computing Research Repository*, 1412.6980, 2015. 3.3.1, B.2, C.5

[47] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55(5), 2014. 4.5, D.2

[48] Thanh Chi Lam, Nghia Hoang, Bryan Kian Hsiang Low, and Patrick Jaillet. Model fusion for personalized learning. In *International Conference on Machine Learning*, pages 5948–5958. PMLR, 2021. 4.3

[49] Y. LeCun, C. Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]*, 2, 2010. URL `http://yann.lecun.com/exdb/mnist`. 4.5, D.2

[50] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015. 1.1, 4.1

[51] Heng Li. Minimap2: Pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–

3100, 2018. 3.1

[52] Jason Liang, Elliot Meyerson, Babak Hodjat, Dan Fink, Karl Mutch, and Risto Miikkulainen. Evolutionary neural automl for deep learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 401–409, 2019. 1.2

[53] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: a big comparison for nas. *arXiv preprint arXiv:1912.06059*, 2019. 1.2

[54] Shih-Wei Lin, Kuo-Ching Ying, Shih-Chieh Chen, and Zne-Jung Lee. Particle swarm optimization for parameter determination and feature selection of support vector machines. *Expert systems with applications*, 35(4):1817–1824, 2008. 1.2

[55] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 4.2.3, 4.2.3

[56] Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sanchez Ramos, and José Ranilla Pastor. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 481–488, 2017. 1.2

[57] X. Lu, J. Gonzalez, Z. Dai, and N. Lawrence. Structured variationally auto-encoded optimization. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3267–3275, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/lu18c.html`. 1.1, 1.2, 1.3, 2.2.2, 2.3, 2.4, 2.5, A.1, A.1, A.2

[58] G. Malkomes, C. Schaff, and R. Garnett. Bayesian optimization for automated model selection. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2900–2908, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9. URL `http://dl.acm.org/citation.cfm?id=3157382.3157422`. 1.1, 1.2, 1.3, 2.2.2, 2.3, 2.4, A.1

[59] Guillaume Marçais, David Pellow, Daniel Bork, et al. Improving the performance of minimizers and winnowing schemes. *Bioinformatics*, 33(14):i110–i117, 2017. 1.2, 1.3, 3.1, 3.2.1, 3.3.1, 5.2, B.1.2, B.1.3

[60] Guillaume Marçais, Dan DeBlasio, and Carl Kingsford. Asymptotically optimal minimizers schemes. *Bioinformatics*, 34(13):i13–i22, 2018. 3.2.1, 3.3.1, 5.2

[61] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proc. AISTATS*, pages 1273–1282, 2017. 1.3, 4.2.3, 4.3, 4.5, D.1.2

[62] Karen H. Miga, Sergey Koren, Arang Rhie, et al. Telomere-to-Telomere assembly of a complete human X chromosome. *Nature*, 585(7823):79–84, 2020. B.2.1, C.4

[63] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*, pages 293–312. Elsevier, 2019. 1.2

[64] Risto Miikkulainen, Olivier Francon, Elliot Meyerson, Xin Qiu, Darren Sargent, Elisa Canzani, and Babak Hodjat. From prediction to prescription: evolutionary optimization of nonpharmaceutical interventions in the covid-19 pandemic. *IEEE Transactions on Evolutionary Computation*, 25(2): 386–401, 2021. 1.2

[65] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017. A.1.4

[66] Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR, 2016. 1.2

[67] Yaron Orenstein, David Pellow, Guillaume Marçais, et al. Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing. *PLOS Computational Biology*, 13: e1005777, 10 2017. 3.2.1

[68] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018. 1.1, 1.2, 4.2.1

[69] P. S. Rana. Physicochemical Properties of Protein Tertiary Structure Data Set, 2013. URL `http://archive.ics.uci.edu/ml/datasets/`. 2.5, A.2, A.2.2

[70] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. 1.2, 2.1, 5.1, A.1.3

[71] John R Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 65–118. Elsevier, 1976. 1.1

[72] Michael Roberts, Wayne Hayes, Brian Hunt, et al. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20:3363–9, 01 2005. 3.1

[73] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 76–85, 2003. 1.3, 3.1, 3.1, 5.2

[74] Jim Shaw and Yun William Yu. Theory of local k-mer selection with applications to long-read alignment. *Bioinformatics*, pages 4659–4669, 2021. ISSN 1367-4803. 1.3, 3.1, 3.4.1, 5.2, C.1.2, C.3

[75] J. Snoek, L. Hugo, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proc. NIPS*, pages 2960–2968, 2012. 1.2, 2.2.2, A.1.3

[76] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. ICML*, pages 1015–1022, 2010. 1.2

[77] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013. 1.1

[78] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017. 1.1, 4.1

[79] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. de Freitas. Bayesian optimization in a billion dimensions via random embeddings. *JAIR*, 55:361–387, 2016. 2.5, A.2, A.2

[80] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

1.2

[81] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018. 4.1, 4.2.3

[82] Chengxi Ye, Zhanshan Sam Ma, Charles H. Cannon, et al. Exploiting sparseness in de novo genome assembly. In *BMC Bioinformatics*, volume 13, pages 1–8. BioMed Central, 2012. 3.1

[83] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, and Nghia Hoang. Statistical model aggregation via parameter matching. *Advances in Neural Information Processing Systems*, 32:10956–10966, 2019. 4.3

[84] Zhengdong D. Zhang, Alberto Paccanaro, Yutao Fu, et al. Statistical analysis of the genomic distribution and correlation of regulatory elements in the encode regions. *Genome Research*, 17(6): 787–797, 2007. 3.2.1

[85] Hongyu Zheng, Carl Kingsford, and Guillaume Marçais. Improved design and analysis of practical minimizers. *Bioinformatics*, 36(Supplement_1):i119–i127, 2020. 1.1, 1.2, 1.3, 3.2.1, 3.3.1, 3.5, B.1.1, B.2.1, B.2.3, B.2.5, C.2, C.3, C.3, C.3

[86] Hongyu Zheng, Carl Kingsford, and Guillaume Marçais. Sequence-specific minimizers via polar sets. *Bioinformatics*, 37:i187–i195, 2021. 1.1, 1.2, 3.2.3, 3.3.1, 3.5, B.1.1, B.2.1, B.2.5, C.2

[87] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 1.1, 1.2, 4.1