Federated Nearest Neighbor Classification with a Colony of Fruit-Flies

Parikshit Ram[1], **Kaushik Sinha**[2]

[1]IBM Research AI, [2]Wichita State University
p.ram@acm.org, kaushik.sinha@wichita.edu

Abstract

The mathematical formalization of a neurological mechanism in the olfactory circuit of a fruit-fly as a locality sensitive hash (`FlyHash`) and bloom filter (`FBF`) has been recently proposed and "reprogrammed" for various machine learning tasks such as similarity search, outlier detection and text embeddings. We propose a novel reprogramming of this hash and bloom filter to emulate the canonical nearest neighbor classifier (`NNC`) in the challenging Federated Learning (FL) setup where training and test data are spread across parties and no data can leave their respective parties. Specifically, we utilize `FlyHash` and `FBF` to create the `FlyNN` classifier, and theoretically establish conditions where `FlyNN` matches `NNC`. We show how `FlyNN` is trained *exactly* in a FL setup with low communication overhead to produce `FlyNN`FL, and how it can be differentially private. Empirically, we demonstrate that (i) `FlyNN` matches `NNC` accuracy across 70 OpenML datasets, (ii) `FlyNN`FL training is highly scalable with low communication overhead, providing up to $8\times$ speedup with 16 parties.

1 Introduction

Biological systems (such a neural networks (Kavukcuoglu et al. 2010; Krizhevsky, Sutskever, and Hinton 2012), convolutions (Lecun and Bengio 1995), dropout (Hinton et al. 2012), attention mechanisms (Larochelle and Hinton 2010; Mnih et al. 2014)) have served as inspiration to modern deep learning systems, demonstrating amazing empirical performance in areas of computer vision, natural language programming and reinforcement learning. Such learning systems are not biologically viable anymore, but the biological inspirations were critical. This has motivated a lot of research into identifying other biological systems that can inspire development of new and powerful learning mechanisms or provide novel critical insights into the workings of intelligent systems. Such neurobiological mechanisms have been identified in the olfactory circuit of the brain in a common fruit-fly, and have been re-used for common learning problems such as similarity search (Dasgupta, Stevens, and Navlakha 2017; Ryali et al. 2020), outlier detection (Dasgupta et al. 2018), and more recently for word embeddings (Liang et al. 2021) and centralized classification (Sinha and Ram 2021). More precisely, in the fruit-fly olfactory circuit, an odor activates a small set of Kenyon Cells (KC) which represent a "tag" for the odor. This tag generation process can be viewed as a natural hashing scheme (Dasgupta, Stevens, and Navlakha 2017), termed `FlyHash`, which generates a high dimensional but very sparse representation (2000 dimensions with 95% sparsity). This tag/hash creates a response in a specific mushroom body output neuron (MBON) – the MBON-$\alpha'3$ – corresponding to the perceived novelty of the odor. Dasgupta et al. (2018) "interpret the KC$\to$MBON-$\alpha'3$ synapses as a Bloom Filter" that creates a "memory" of all the odors encountered by the fruit-fly, and reprogram this *Fly Bloom Filter* (`FBF`) as a novelty detection mechanism that performs better than other locality sensitive Bloom Filter-based novelty detectors for neural activity and vision datasets.

We build upon the reprogramming of the KC$\to$MBON-$\alpha'3$ synapses as the `FBF` to create a supervised classification scheme. We show that this classifier mimics a nearest-neighbor classifier (`NNC`). This scheme possesses several unique desirable properties that allows for nearest-neighbor classification in the federated learning (FL) setup with a low communication overhead. In FL setup the complete training data is distributed across multiple parties and none of the original data (training or testing) is to be exchanged between the parties. This is possible because of the unique high-dimensional sparse structure of the `FlyHash`. We consider this an exercise of leveraging "naturally occurring" algorithms to solve common learning problems (which these natural algorithms were not designed for), resulting in schemes with unique capabilities. Nearest neighbor classification (`NNC`) is a fundamental nonparametric supervised learning scheme, with various theoretical guarantees and strong empirical capabilities (especially with an appropriate similarity function). FL has gained a lot of well-deserved interest in the recent years as, on one hand, models become more data hungry, requiring data to be pooled from various sources, while on the other hand, ample focus is put on data privacy and security, restricting the transfer of data. However, the very nature of `NNC` makes it unsuitable for FL – for any test point at a single party, obtaining the nearest neighbors would *naively* either require data from all parties to be collected at the party with the test point, or require the test point to be sent to all parties to obtain the per-party neighbors; both these options violate the desiderata of FL.

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

We leverage the ability of the FBF to summarize a data distribution in a bloom filter to develop a classifier where every class is summarized with its own FBF, and inference involves selecting the class whose distribution (represented by its own FBF) is most similar to the test point. We theoretically and empirically show that this classifier, which we name FlyNN (Fly Nearest Neighbor) classifier, approximately agrees with NNC. We then perform NNC with FlyNN on distributed data under the FL setup with low communication overhead. The key idea is to train a FlyNN separately on each party – that is, have a "colony of fruit-flies" – and then perform a low communication aggregation at training time without having to exchange any of the original data. This enables low communication federated nearest-neighbor classification with FlyNNFL. One unique capability enabled by this neurobiological mechanism is that FlyNNFL can perform NNC without transferring the test point to other parties in any form. We make the following contributions[1]:

- We present the FlyNN classifier utilizing the FBF and FlyHash, and theoretically present precise conditions under which FlyNN matches the NNC.
- We present an algorithm for training FlyNN with distributed data in the FL setting, with low communication overhead and differential privacy, without requiring exchange of the original data.
- We empirically compare FlyNN to NNC and other relevant baselines on 70 classification datasets from the OpenML (Van Rijn et al. 2013) data repository.
- We demonstrate the scaling of the data distributed FlyNN training on datasets of varying sizes to highlight the low communication overhead of the proposed scheme.

The paper is organized as follows: We detail the FlyNN classifier and analyze its theoretical properties in §2. We present federated kNNC via distributed FlyNN in §3. We empirically evaluate our proposed methods against baselines in §4, discuss related work in §5 and conclude with a discussion on limitations and future work in §6. The theoretical proofs and implementation details are presented in the supplement of a separate extended version (Ram and Sinha 2021a).

2 The FlyNN Classifier

In our presentation, we use lowercase letters (x) for scalars and functions (with arguments), boldface lowercase letters ($\boldsymbol{x}$) for vectors, lowercase SansSerif letter (h) for Booleans, boldface lowercase SansSerif letter ($\mathbf{h}$) for Boolean vectors, and uppercase SansSerif letter (M) for Boolean matrices. For a vector $\boldsymbol{x}$, $\boldsymbol{x}[j]$ denotes its j^{th} index. For any positive integer $k \in \mathbb{N}$, we use $[k]$ to denote the set $\{1, \ldots, k\}$.

We start this section by recalling k-nearest neighbor classification (kNNC). Given a dataset of labeled points $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times [L]$ from L classes, and a similarity function $s : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$, a test point $\boldsymbol{x} \in \mathbb{R}^d$ is labeled by the kNNC based on its k-nearest neighbors $S^k(\boldsymbol{x}) = \operatorname{argmax}_{R \subset S \colon |R|=k} \sum_{(\boldsymbol{x}_i, y_i) \in R} s(\boldsymbol{x}, \boldsymbol{x}_i)$ as:

$$\hat{y} \leftarrow \operatorname*{argmax}_{y \in [L]} \left| \left\{ (\boldsymbol{x}_i, y_i) \in S^k(\boldsymbol{x}) : y_i = y \right\} \right|, \qquad (1)$$

In the federated version of kNNC, the data is distributed across τ parties, each with a chunk of the data $S_t, t \in [\tau]$. For a test point $\boldsymbol{x}$ at a specific party t_{in}, the classification should be based on the nearest-neighbors of $\boldsymbol{x}$ over the pooled data $S_1 \cup S_2 \cdots \cup S_\tau$. A critical desiderata in FL is to be robust to the fact that the per-party data S_t are not obtained from identical distributions for all $t \in [\tau]$ – the distributions that generate S_t and $S_{t'}$ for $t \neq t'$ can be significantly different. We refer to this as the "non-IID-ness of the per-party data".

We leverage the locality sensitive FlyHash (Dasgupta, Stevens, and Navlakha 2017) in our proposed scheme, focusing on the binarized version (Dasgupta et al. 2018). For $\boldsymbol{x} \in \mathbb{R}^d$, the FlyHash $h \colon \mathbb{R}^d \to \{0,1\}^m$ is defined as,

$$h(\boldsymbol{x}) = \Gamma_\rho(\mathsf{M}\boldsymbol{x}), \qquad (2)$$

where $\mathsf{M} \in \{0,1\}^{m \times d}$ is the randomized sparse lifting binary matrix with $s \ll d$ nonzero entries in each row, and $\Gamma_\rho \colon \mathbb{R}^m \to \{0,1\}^m$ is the winner-take-all function converting a vector in $\mathbb{R}^m$ to one in $\{0,1\}^m$ by setting the highest $\rho \ll m$ elements to 1 and the rest to zero. FlyHash projects up or *lifts* the data dimensionality ($m \gg d$).

The *Fly Bloom Filter* (FBF) $\boldsymbol{w} \in (0,1)^m$ summarizes a dataset and is subsequently used for novelty detection (Dasgupta et al. 2018) with novelty scores for any point $\boldsymbol{x}$ proportional to $\boldsymbol{w}^\top h(\boldsymbol{x})$ – higher values indicate high novelty of $\boldsymbol{x}$. To learn $\boldsymbol{w}$ from a set S, all its elements are initially set to 1. For an "inlier" point $\boldsymbol{x}_{\mathsf{in}} \in S$ with FlyHash $\mathbf{h}_{\mathsf{in}}$, $\boldsymbol{w}$ is updated by "decaying" (with a multiplicative factor) the intensity of the elements in $\boldsymbol{w}$ corresponding to the nonzero elements in $\mathbf{h}_{\mathsf{in}}$. This ensures that some $\boldsymbol{x} \approx \boldsymbol{x}_{\mathsf{in}}$ receives a low novelty score $\boldsymbol{w}^\top h(\boldsymbol{x})$. For a novel point $\boldsymbol{x}_{\mathsf{nv}}$ (with FlyHash $\mathbf{h}_{\mathsf{nv}}$) not similar to any $\boldsymbol{x} \in S$, the locality sensitivity of FlyHash implies that, with high probability, the elements of $\boldsymbol{w}$ corresponding to the nonzero elements in $\mathbf{h}_{\mathsf{nv}}$ will be close to 1 since their intensities will not have been decayed much, implying a high novelty score $\boldsymbol{w}^\top \mathbf{h}_{\mathsf{nv}}$.

2.1 FlyNN Algorithm: Training and Inference

We leverage this mechanism for classification by using the FBF to summarize each class $l \in [L]$ separately – the per-class FBF encodes the local neighborhoods of each class, and the high dimensional sparse nature of FlyHash (and consequently FBF) summarizes classes with multi-modal distributions while reducing inter-class FBF overlap.

FlyNN training. Given a training set $S \subset \mathbb{R}^d \times [L]$, the learning of the per-class FBFs $\boldsymbol{w}_l \in (0,1)^m, l \in [L]$ is detailed in the TrainFlyNN subroutine in Algorithm 1. We initialize the FlyHash by randomly generating the M (line 2). The per-class FBF $\boldsymbol{w}_l$ are initialized to $\mathbf{1}_m$ (line 3). For a training example $(\boldsymbol{x}, y) \in S$, we first generate the FlyHash $\mathbf{h} = h(\boldsymbol{x}) \in \{0,1\}^m$ using equation 2 (line 5). Then, the FBF $\boldsymbol{w}_y$ (corresponding to $\boldsymbol{x}$'s class y) is updated with the FlyHash $\mathbf{h}$ as follows – the elements of $\boldsymbol{w}_y$ corresponding to the nonzero bit positions of $\mathbf{h}$ are decayed, and the rest of the entries of $\boldsymbol{w}_y$ are left as is (line 6); the remaining FBFs $\boldsymbol{w}_l, l \neq y \in [L]$ are not updated at all. The decay is achieved by multiplication with a factor of $\gamma \in [0,1)$ – large γ implies slow decay in the FBF intensity; a small value of

[1] A preliminary version of this paper was presented at a recent workshop (Ram and Sinha 2021b).

Algorithm 1: FlyNN training with training set $S \subset \mathbb{R}^d \times [L]$, lifted dimensionality $m \in \mathbb{N}$, $s \ll d$ nonzeros in each row of the lifting matrix M, $\rho \ll m$ nonzeros in the FlyHash, decay rate $\gamma \in [0, 1)$, random seed R, and inference with test point $\boldsymbol{x} \in \mathbb{R}^d$.

```
TrainFlyNN($S, m, \rho, s, \gamma, R$)
    Sample $\mathsf{M} \in \{0,1\}^{m \times d}$ with seed $R$
    Initialize $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_L \leftarrow \mathbf{1}_m \in (0,1)^m$
    for $(\boldsymbol{x}, y) \in S$ do
        $\mathbf{h} \leftarrow \Gamma_\rho(\mathsf{M}\boldsymbol{x})$
        $\boldsymbol{w}_y[i] \leftarrow \gamma \cdot \boldsymbol{w}_y[i] \; \forall i \in [m] \colon \mathbf{h}[i] = 1$
    end
    return $(\mathsf{M}, \{\boldsymbol{w}_l, l \in [L]\})$
end
InferFlyNN($\boldsymbol{x}, \mathsf{M}, \rho, \{\boldsymbol{w}_l, l \in [L]\}$)
    $\mathbf{h} \leftarrow \Gamma_\rho(\mathsf{M}\boldsymbol{x})$
    return $\operatorname{argmin}_{l \in [L]} \boldsymbol{w}_l^\top \mathbf{h}$
end
```

γ triggers rapid decay ($\gamma = 0$ makes the FBFs binary). This whole process ensures that $\boldsymbol{x}$ (and points similar to $\boldsymbol{x}$) are considered to be an "inlier" with respect to $\boldsymbol{w}_y$.

FlyNN inference. The FBF $\boldsymbol{w}_l$ for class $l \in [L]$ are learned such that a point $\boldsymbol{x}$ with label l appears as an inlier with respect to $\boldsymbol{w}_l$ (class l); the example $(\boldsymbol{x}, y)$ does not affect the other class FBFs $\boldsymbol{w}_l, l \neq y, l \in [L]$. This implies that a point $\boldsymbol{x}'$ similar to $\boldsymbol{x}$ will have a low novelty score $\boldsymbol{w}_y^\top h(\boldsymbol{x}')$ motivating our inference rule – for a test point $\boldsymbol{x}$, we compute the per-class novelty scores and predict the label as $\hat{y} \leftarrow \arg\min_{l \in [L]} \boldsymbol{w}_l^\top h(\boldsymbol{x})$. This is detailed in the InferFlyNN subroutine in Algorithm 1.

2.2 Analysis of FlyNN

We first present the computational complexities of Algorithm 1 for a specific configuration of its hyper-parameters. All proofs are presented in Ram and Sinha (2021a, Appendix A).

Lemma 1 (FlyNN training)**.** *Given a training set $S \subset \mathbb{R}^d \times [L]$ with n examples, the TrainFlyNN subroutine in Algorithm 1 with the lifted FlyHash dimensionality m, number of nonzeros s in each row of $\mathsf{M} \in \{0,1\}^{m \times d}$, number of nonzeros ρ in FlyHash $h(\boldsymbol{x})$ for any $\boldsymbol{x} \in \mathbb{R}^d$, and decay rate $\gamma \in [0, 1)$ takes time $O(nm \cdot \max\{s, \log \rho\})$ and has a memory overhead of $O(m \cdot \max\{s, L\})$.*

Lemma 2 (FlyNN inference)**.** *Given a trained FlyNN, the inference for $\boldsymbol{x} \in \mathbb{R}^d$ with the InferFlyNN subroutine in Algorithm 1 takes time $O\left(m \cdot \max\left\{s, \log \rho, {}^{\rho L}\!/_{m}\right\}\right)$ with a memory overhead of $O(\max\{m, L\})$.*

Remark 1. *For any test point $\boldsymbol{x} \in \mathbb{R}^d$ with FlyHash $h(\boldsymbol{x})$, and a large number of labels (large L), if the $\arg\min_{l \in [L]} \boldsymbol{w}_l^\top h(\boldsymbol{x})$ can be solved via* **fast maximum inner product search** *(Koenigstein, Ram, and Shavitt 2012; Ram and Gray 2012) in time $\beta(L)$ sublinear[2] in L, then the overall inference time for $\boldsymbol{x}$ would be given by $O\left(m \cdot \max\left\{s, \log \rho, {}^{\rho\beta(L)}\!/_{m}\right\}\right)$ which is sublinear in L.*

Next we present learning theoretic properties of FlyNN. The novelty score $\boldsymbol{w}_l^\top h(\boldsymbol{x})$ of any test point $\boldsymbol{x}$ in FlyNN corresponds to how "far" $\boldsymbol{x}$ is from the distribution of class l encoded by $\boldsymbol{w}_l$, and using the class with the minimum novelty score to label $\boldsymbol{x}$ is equivalent to labeling $\boldsymbol{x}$ with the class whose distribution is "closest" to $\boldsymbol{x}$. With this intuition, we identify precise conditions where FlyNN mimics kNNC. All proofs are presented in Ram and Sinha (2021a, Appendix B).

We present our analysis for binary classification with $\gamma = 0$, where the FlyNN is trained on training set $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \{0, 1\}$. Let $S^0 = \{(x, y) \in S : y = 0\}$, $S^1 = \{(x, y) \in S : y = 1\}$ and let $\boldsymbol{w}_0, \boldsymbol{w}_1 \in \{0, 1\}^m$ be the FBFs constructed using S^0 and S^1 respectively. Without loss of generality, for any test point $\boldsymbol{x}$, assume that the majority of its k nearest neighbors from S has class label 1. Thus kNNC will predict $\boldsymbol{x}$'s class label to be 1. We aim to show that $\mathbb{E}_{\mathsf{M}}(\boldsymbol{w}_1^\top h(\boldsymbol{x})) < \mathbb{E}_{\mathsf{M}}(\boldsymbol{w}_0^\top h(\boldsymbol{x}))$ (expectation of the random M matrix) so that FlyNN will predict, in expectation, $\boldsymbol{x}$'s label to be 1. A high probability statement will then follow using standard concentration bounds. If $\boldsymbol{x}$'s nearest neighbor is arbitrarily close to $\boldsymbol{x}$ and has label 0 (while the label of the majority of its k nearest neighbors still being 1) then we would expect $\boldsymbol{w}_0^\top h(\boldsymbol{x}) < \boldsymbol{w}_1^\top h(\boldsymbol{x})$ with high probability, thereby, FlyNN will label $\boldsymbol{x}$ as 0. To avoid such a situation, we assume a margin $\eta > 0$ between the classes (Gottlieb, Kontorovich, and Nisnevitch 2014) defined as:

Definition 1. *We define the* margin η *of the training set S to be $\eta \triangleq \min_{\boldsymbol{x} \in S^0, \boldsymbol{x}' \in S^1} \|\boldsymbol{x} - \boldsymbol{x}'\|_\infty$.*

If $\lceil {}^{k+1}\!/_{2} \rceil$ of $\boldsymbol{x}$'s nearest neighbors from S are at a distance at most $\eta/2$ from $\boldsymbol{x}$, then all of those $\lceil {}^{k+1}\!/_{2} \rceil$ examples must have the same class label to which kNNC agrees. This also ensures that the closest point to $\boldsymbol{x}$ from S having opposite label is at least $\eta/2$ distance away. We show next that this is enough to ensure that prediction of FlyNN on any test point $\boldsymbol{x}$ from a ***permutation invariant*** distribution agrees with the prediction of kNNC with high probability (the training set can be from any distribution). Note that P is a permutation invariant distribution over $\mathbb{R}^d$ if for any permutation σ of $[d]$ and any $\boldsymbol{x} \in \mathbb{R}^d$, $P(x_1, \ldots, x_d) = P(x_{\sigma(1)}, \ldots, x_{\sigma(d)})$.

Theorem 3. *Fix s, ρ, m and k. Given a training set S of size n and a test example $\boldsymbol{x} \in \mathbb{R}^d$ sampled from a permutation invariant distribution, let $\boldsymbol{x}_*$ be its $\left(\lceil {}^{k+1}\!/_{2} \rceil\right)^{th}$ nearest neighbor from S measured using ℓ_∞ metric. If $\|\boldsymbol{x} - \boldsymbol{x}_*\|_\infty \leq \min\{{}^{\eta}\!/_{2}, O({}^{1}\!/_{s})\}$ then, $\hat{y}_{\mathtt{FlyNN}} = \hat{y}_{k\mathtt{NNC}}$ with probability $\geq 1 - \left(O({}^{\rho n}\!/_{m}) + e^{-O(\rho)}\right)$, where $\hat{y}_{\mathtt{FlyNN}}$ and $\hat{y}_{k\mathtt{NNC}}$ are respectively the predictions of FlyNN and kNNC.*

Remark 2. *For any $\delta \in (0, 1)$, the failure probability of the above theorem can be restricted to at most δ by setting $\rho = \Omega(\log {}^{1}\!/_{\delta})$ and $m = \Omega({}^{n\rho}\!/_{\delta})$.*

Remark 3. *We established conditions under which the predictions of FlyNN agrees with that of kNNC with high probability. kNNC is a non-parametric classification method with strong theoretical guarantee: as $|S| = n \to \infty$, the kNNC almost surely approaches the Bayes optimal error rate. Therefore, by establishing the connection between FlyNN and*

[2] For example, $\beta(L) \sim O(\log L)$ using randomized partition trees (Keivani, Sinha, and Ram 2017, 2018) or cover trees (Curtin, Ram, and Gray 2013).

Algorithm 2: Federated Differentially Private `FlyNN` training with τ parties $V_t, t \in [\tau]$ each with training set S_t with DP parameters ϵ and number of samples T. The Boolean `IS_DP` toggles DP.

```
1  TrainFlyNNFLDP($\{S_t, t \in [\tau]\}, m, \rho, s, \gamma, \texttt{IS\_DP}, \epsilon, T$)
2      Generate random seed $R$ & broadcast to all $V_t, t \in [\tau]$
3      for each party $V_t, t \in [\tau]$ do
4          $(\mathsf{M}, \{\boldsymbol{w}_l^t, l \in [L]\}) \leftarrow$ TrainFlyNN $(S_t, m, \rho, s, \gamma, R)$
5          if IS_DP then
6              $\{\boldsymbol{w}_l^t, l \in [L]\} \leftarrow \mathrm{DP}((\{\boldsymbol{w}_l^t, l \in L\}, \epsilon/\tau, T))$
7          end
8      end
       // All-reduction over all $\tau$ parties
9      $\hat{\boldsymbol{w}}_l[i] \leftarrow \gamma^{\sum_{t \in [\tau]} \log_\gamma \boldsymbol{w}_l^t[i]} \forall i \in [m], \forall l \in [L]$
10     return $(\mathsf{M}, \{\hat{\boldsymbol{w}}_l, l \in [L]\})$ on each party $V_t, t \in [\tau]$
11 end
12 DP($\{\boldsymbol{w}_l, l \in L\}, \epsilon, T$)
13     $\boldsymbol{w} \leftarrow [(\boldsymbol{w}_1)^\top, \ldots, (\boldsymbol{w}_L)^\top]^\top$
14     $\boldsymbol{c}[i] \leftarrow \log_\gamma \boldsymbol{w}[i] \; \forall i \in [m \times L]$
15     $\mathcal{R} \leftarrow \{\}$
16     for $j \in [T]$ do
17         Sample $i_j \in [m \times L]$ with probability $\propto \exp(\epsilon \boldsymbol{c}[i_j]/4T)$
18         $\boldsymbol{c}[i_j] \leftarrow \max\{\boldsymbol{c}[i_j] + \eta, 0\}$, where $\eta \sim \mathsf{Laplace}(2T/\epsilon)$
19         $\mathcal{R} \leftarrow \mathcal{R} \cup \{i\}$
20     end
21     $\boldsymbol{c}[i] \leftarrow 0, \forall i \in [m \times L] \setminus \mathcal{R}$
22     $\tilde{\boldsymbol{w}}[i] \leftarrow \gamma^{\boldsymbol{c}[i]}, \forall i \in [m \times L]$
23     $[(\tilde{\boldsymbol{w}}_1)^\top, \ldots, (\tilde{\boldsymbol{w}}_L)^\top]^\top \leftarrow \tilde{\boldsymbol{w}}$
24     return $(\{\tilde{\boldsymbol{w}}_l, l \in [L]\})$
25 end
```

kNNC, `FlyNN` has the same statistical guarantee under the conditions of Theorem 3.

Remark 4. *For $k = 1$, prediction of 1NNC on any $\boldsymbol{x} \in \mathbb{R}^d$ agrees with the label of its nearest neighbor $\boldsymbol{x}'$ and any point in the training set having class label different from the label of $\boldsymbol{x}'$ is farther away from $\boldsymbol{x}$. Here we do not need any dependence on margin η and the condition $\|\boldsymbol{x} - \boldsymbol{x}'\|_\infty = O(1/s)$ is enough to get a statement similar to Theorem 3 that relate `FlyNN` to 1NNC.*

3 Federated NNC via Distributed FlyNN

For federated learning where the data S is spread across τ parties with each party $V_t, t \in [\tau]$ having its own chunk S_t, we present a distributed `FlyNN`FL learning scheme in Algorithm 2, highlighting the differences from the original `FlyNN` learning in Blue text. The Boolean `IS_DP` toggles the differential privacy (DP) of the training. This scheme ensures *inter-party privacy*, protecting against leakage even with colluding parties. The proofs for the analyses in this section are presented in Ram and Sinha (2021a, Appendix C).

In Train`FlyNN`FLDP, all the parties $V_t, t \in [\tau]$ have the complete `FlyNN` model at the conclusion of the training, and are *able to perform no-communication inference on any new test point $\boldsymbol{x}$ independent of the other parties* using the Infer`FlyNN` subroutine in Algorithm 1. The learning commences by generating and broadcasting a random seed R to all parties $V_t, t \in [\tau]$ (line 2); we assume that all parties already have knowledge of the total number of labels L. Then each party V_t independently invokes Train`FlyNN` (Algorithm 1) on its chunk S_t and obtains the per-class private or non-private `FBF` $\{\boldsymbol{w}_l^t, l \in [L]\}$ depending on the status of the Boolean variable `IS_DP` and the invocation of the DP subroutine (lines 3-8). Finally, a specialized *all-reduce* aggregates all the per-class `FBF`s $\{\boldsymbol{w}_l^t, l \in [L]\}$ across all parties $t \in [\tau]$ to obtain the final `FBF`s $\hat{\boldsymbol{w}}_l, l \in [L]$ on all parties (line 9). In the DP module, the input `FBF` $\{\boldsymbol{w}_l, l \in [L]\}$ are concatenated and the element-wise log values (counts) are stored in a vector $\boldsymbol{c}$ (lines 13-14). Then the largest T indices of c are selected iteratively using an exponential mechanism and Laplace noise are added to these selected entries (lines 16-20). The remaining $(m \times L) - T$ entries of $\boldsymbol{c}$ are set to zero (line 21), all the entries of $\boldsymbol{c}$ are exponentiated and the differentially private `FBF` $\{\tilde{\boldsymbol{w}}_l^t, l \in [L]\}$ are returned (lines 22-24). The following claim establishes exact parity between the non-DP federated and original training of `FlyNN`:

Theorem 4 (Non-private Federated training parity)**.** *Given training sets $S_t \subset \mathbb{R}^d \times [L]$ on each party $V_t, t \in [\tau]$, and a `FlyNN` configured as in Lemma 1, if the Boolean variable* `IS_DP` *is False, then the per-party final `FlyNN` $\{\hat{\boldsymbol{w}}_l, l \in [L]\}$ (Algorithm 2, line 9) output by Train`FlyNN`FLDP $(\{S_t, t \in [\tau]\}, m, s, \rho, \gamma, \texttt{IS\_DP}, \epsilon, T)$ with random seed R in Algorithm 2 is equal to the `FlyNN` $\{\boldsymbol{w}_l, l \in [L]\}$ (Algorithm 1, line 8) output by Train`FlyNN` (S, m, s, ρ, c, R) subroutine in Algorithm 1 with the pooled training set $S = \cup_{t \in [\tau]} S_t$.*

This implies that the `FlyNN`FL training (i) *does not incur any approximation*, **and** (ii) *does not require any original training data to leave their respective parties*, and these aggregated per-class `FBF`s are now available on every party V_t and used to (iii) *perform inference on test points on each party with no communication to other parties* using the Infer`FlyNN` subroutine in Algorithm 1. The unique capabilities are enabled by the learning dynamics of the `FBF` in `FlyNN`.

Remark 5 (Agnostic to non-IID-ness of per-party data)**.** *Theorem 4 implies that the proposed `FlyNN`FL is completely agnostic to the non-IID-ness of the data across parties. The proposed scheme approximates the ideal kNNC (which has unrestricted access to data from all the parties)* **regardless of the non-IID-ness of the per-party data**.

The computational complexities of `FlyNN`FL training are as follows:

Lemma 5 (`FlyNN`FL training)**.** *Given the setup in Theorem 4 with $|S_t| = n_t$, Train`FlyNN`FLDP (Algorithm 2) takes $O(m \cdot \max\{s, L\})$ memory, with (a) $O(n_t m \cdot \max\{s, \log \rho, L/n_t \log \tau\})$ time per-party and $O(mL\tau)$ communication with DP disabled (`IS_DP`=false), and (b) $O(n_t m \cdot \max\{s, \log \rho, LT/n_t\} + T \log \tau)$ time per-party and $O(T\tau)$ communication with DP enabled.*

The following result establishes the DP property of Train`FlyNN`FLDP, which prevents leakage between parties during the training procedure. The proof leverages the exponential mechanism.

Theorem 6. *With the DP module enable (`IS_DP`=true), Train`FlyNN`FLDP is $(\epsilon, 0)$ differentially private.*

Communication setup. The TrainFlyNNFLDP algorithm is presented here in a peer-to-peer communication setup. However, it will easily transfer to a centralized setup with a "global aggregator" that all parties communicate to. In that case, for FlyNNFL training, the aggregator (i) generates and broadcasts the seed, and (ii) gathers & computes $\{\hat{\boldsymbol{w}}_l, l \in [L]\}$, and (iii) broadcasts them to all parties.

Effect of timed-out parties ("Stragglers") in FL. The ability to be robust to stragglers is of critical importance in FL. Stragglers play an important role in iterative algorithms with multiple rounds of communication. In TrainFlyNNFLDP, there is only *a single round of communication* in the training scheme (Algorithm 2, line 9), we do not anticipate there to be any stragglers. For inference, *all computations are local to each party*, and hence, there is no notion of stragglers. We leave further study of stragglers for future work.

4 Empirical Evaluation

In this section, we evaluate the empirical performance of FlyNN. First, we compare FlyNN to NNC to validate its ability to approximate NNC. Then, we demonstrate the scaling of FlyNNFL training on data distributed among multiple parties. Finally, we present the privacy-performance tradeoff of FlyNNFL. Various details and additional experiments are presented in Ram and Sinha (2021a, Appendix D). The implementation details and compute resources used are described in Ram and Sinha (2021a, Appendix D.1) and relevant code is available at https://github.com/rithram/flynn.

Datasets. For the evaluation of FlyNN, we consider three groups of datasets:

- We consider binary and continuous **synthetic data** of varying sizes, designed to favor local classifiers like NNC (Guyon 2003). See Ram and Sinha (2021a, Appendix D.2) for further details.
- We consider 70 classification datasets from OpenML (Van Rijn et al. 2013) to evaluate the performance of FlyNN on real datasets, **thoroughly** comparing FlyNN to NNC. See Ram and Sinha (2021a, Appendix D.3) for details.
- We consider high dimensional vision datasets MNIST (Lecun 1995), Fashion-MNIST (Xiao, Rasul, and Vollgraf 2017) and CIFAR (Krizhevsky, Hinton et al. 2009) from the Tensorflow package (Abadi et al. 2016) for evaluating the scaling of FlyNNFL training when the data is distributed between multiple parties. See Ram and Sinha (2021a, Appendix D.4) for details.

Baselines and ablation. We compare our proposed FlyNN to two baselines:

- *k***NNC:** This is the primary baseline. We tune over the neighborhood size $k \in [1, 64]$. We also specifically consider 1NNC ($k = 1$).
- **SBFC:** To ablate the effect of the high level of sparsity in FlyHash, we utilize the binary SimHash (Charikar 2002) based locality sensitive bloom filter for each class in place of FBF to get SimHash Bloom Filter classifier (SBFC). See Ram and Sinha (2021a, Appendix D.5) for further details.

Table 1: Comparison of FlyNN with NNC and SBFC on synthetic data. We report normalized accuracy aggregated over 30 random synthetic datasets. Normalized accuracy for kNNC is zero hence elided from the results.

$\mathcal{X}$	n	d	1NNC	SBFC	FlyNN
$\{0,1\}^d$	10^3	50	0.11 ± 0.05	0.18 ± 0.04	-0.05 ± 0.02
$\{0,1\}^d$	10^4	50	0.18 ± 0.02	0.51 ± 0.02	-0.03 ± 0.01
$\{0,1\}^d$	10^3	100	0.07 ± 0.03	0.58 ± 0.03	-0.04 ± 0.02
$\mathbb{R}^d$	10^3	50	0.09 ± 0.03	0.68 ± 0.01	-0.07 ± 0.02
$\mathbb{R}^d$	10^4	50	0.11 ± 0.00	0.78 ± 0.00	0.07 ± 0.02
$\mathbb{R}^d$	10^3	100	0.11 ± 0.03	0.66 ± 0.02	-0.05 ± 0.03

FlyNN hyper-parameter search. For a dataset with d dimensions, we tune across 60 FlyNN hyper-parameter settings in the following ranges: $m \in [2d, 2048d]$, $s \in [2, \lfloor 0.5d \rfloor]$, $\rho \in [8, 256]$, and $\gamma \in [0, 0.8]$. We use this hyper-parameter space for all experiments, except for the vision sets, where we use $m \in [2d, 1024d]$. We present various experiments and detailed discussions on the hyper-parameter dependence in Ram and Sinha (2021a, Appendix D.6). To summarize the dependence, (i) increasing m improves FlyNN performance and can be selected to be as large as computationally feasible, (ii) when d is large enough (≥ 20), the FlyNN performance is somewhat agnostic to the choice of s and any small value ($s \sim 0.05d$) suffices, (iii) increasing ρ improves FlyNN performance up to a point after which it can hurt performance unless m is increased as well since it reduces the sparsity of FlyHash, (iv) increasing γ from 0 to > 0 significantly improves FlyNN performance, but otherwise the performance is quite robust to its precise choice.

Evaluation metric to compare across datasets. To obtain statistical significance and error bars for performance across different datasets, we compute the "normalized accuracy" for a method on a dataset as $(1 - {}^{a}/_{a_k})$ where a_k is the best tuned 10-fold cross-validated accuracy of kNNC on this dataset and a is the best tuned 10-fold cross-validated accuracy obtained by the method on this dataset. Thus kNNC has a normalized accuracy of 0 for all datasets; negative values denote improvement over kNNC. This "normalization" allows comparison of the aggregate performance of different methods across different datasets with distinct best achievable accuracies.

Synthetic data. We first consider synthetic data designed for strong kNNC performance. We generate data for 5 classes with 3 clusters per class, and points in the same cluster belong to the same class implying that a neighborhood based classifier will perform well. However, the classes are not linearly separable. We select such a set to demonstrate that the proposed FlyNN is able to encode multiple separate modes of a class within a single FBF while providing enough separation between the per-class FBFs for high predictive performance. We consider binary synthetic data in $\{0,1\}^d$ and synthetic data in general $\mathbb{R}^d$. We consider $d = 50, 100$ and $n = 10^3, 10^4$. For each configuration, we create 30 datasets. The performances of all baselines are presented in Table 1. The results indicate that FlyNN is able to match kNNC perfor-

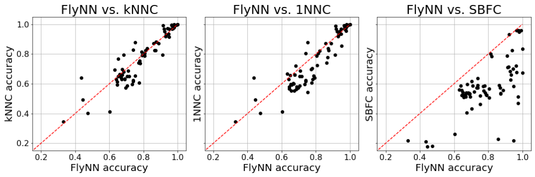


Figure 1: Performance of baselines relative to FlyNN on *OpenML* datasets. The scatter plots compare the best tuned FlyNN accuracy against that of kNNC, 1NNC and SBFC with a point for each dataset, and the red dashed diagonal marking match to FlyNN accuracy.

Table 2: Evaluating FlyNN on OpenML datasets with (i) Fraction of datasets FlyNN outperforms baselines, (ii) Number of datasets FlyNN has wins(W)/ties(T)/losses(L) over baselines, (iii) Median improvement in normalized accuracy by FlyNN over baseline, (iv) p-values for the paired 2-sided t-test (TT), (v) p-values for the 2-sided Wilcoxon signed rank test (WSRT).

METHOD	(i) FRAC.	(ii) W/T/L	(iii) IMP.	(iv) TT	(v) WSRT
kNNC	0.55	39/2/30	0.35%	5.30E-2	7.63E-2
1NNC	0.66	47/2/22	2.36%	1.55E-5	2.81E-5
SBFC	0.99	70/0/1	25.4%	<1E-8	<1E-8

mance significantly better than all other baselines, including 1NNC, by being closest to zero (FlyNN appears to improve upon NNC but the improvements are not significant overall). FlyNN significantly outperforms SBFC, highlighting the need for *sparse high dimensional hashes* to summarize multi-modal distributions while avoiding overlap between per-class FBFs. The small standard errors indicate the stability of the relative performances across different datasets.

OpenML data. We consider 70 classification (binary and multi-class) datasets from OpenML with d numerical columns and n samples; $d \in [20, 1000], n \in [1000, 20000]$. Unlike the synthetic sets, these datasets do not guarantee strong kNNC performance. The results are presented in Figure 1. In Table 2, the normalized accuracy of all baselines are compared to FlyNN with paired two-sided t-tests (TT) and two-sided Wilcoxon signed rank test (WSRT). In Figure 1, we can see on the left figure (kNNC vs FlyNN) that most points are near the diagonal (implying kNNC and FlyNN parity) with some under (better FlyNN accuracy) and some over (worse FlyNN accuracy). With 1NNC in the center plot of Figure 1, we see that, in most cases, 1NNC either matches FlyNN or does worse (being under the diagonal) since kNNC subsumes 1NNC. But the right plot for SBFC in Figure 1 indicates that SBFC is quite unable to match FlyNN (and hence kNNC). We quantify these behaviours in Table 2. FlyNN performs comparably to kNNC (median improvement of only 0.35%) with p-values of 0.0536 (TT) and 0.0763 (WSRT), while improving the normalized accuracy over 1NNC by a median of around 2.36% across all 70 sets (p-values $\sim 10^{-5}$). These results demonstrate that *the proposed FlyNN has comparable performance to properly tuned kNNC and this behaviour is verified with a large number of datasets*. FlyNN significantly outperforms SBFC ($> 25\%$ median improvement, p-values $< 10^{-8}$), again highlighting the value of high sparsity in the FlyHash on real datasets.

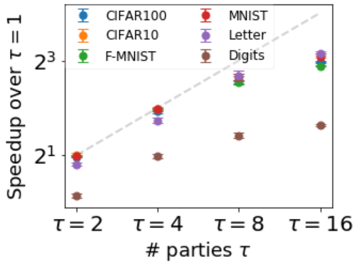
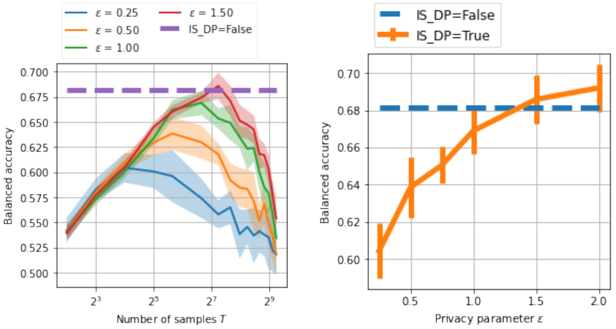


Figure 2: Scaling of FlyNNFL training with τ parties over single-party training. The gray line marks linear scaling.



(a) Effect of T for different ϵ.

(b) Dependence on ϵ.

Figure 3: The effect of the DP parameters T and ϵ on the performance of FlyNNFL (higher is better).

Methods learned through gradient-descent (such as linear models or neural networks) have been widely studied in the FL setting. However, it is hard to compare nearest-neighbor methods against gradient-descent-based methods with proper parity. We present one comparison on these OpenML datasets in Ram and Sinha (2021a, Appendix D.7).

Scaling. We evaluate the scaling of the FlyNNFL training – Algorithm 2, TrainFlyNNFLDP – with the number of parties τ. For fixed hyper-parameters, we average runtimes (and speedups) over 10 repetitions for each of the 6 datasets (see Ram and Sinha (2021a, Appendix D.4)) and present the results in Figure 2. The results indicate that TrainFlyNNFLDP scales very well for up to 8 parties for the larger datasets, and shows up to 8× speed up with 16 parties. There is significant gain (up to 2×) even for the tiny DIGITS dataset (with $<$ 2000 total rows), demonstrating the scalability of the FlyNN training with very low communication overhead.

Differential privacy. To study the privacy-performance tradeoff of FlyNN, we again consider the previously described synthetic data in $\mathbb{R}^d$. For a fixed setting FlyNN hyper-parameters (namely m, s, ρ, γ) and 2-party federated training ($\tau = 2$), we study the effect of the privacy parameters on FlyNNFL performance in Figure 3. In Figure 3a, we see the effect of varying the number of sampled entries T. We observe the intuitive behaviour where, for a fixed privacy level ϵ, increasing T initially improves performance, but eventually hurts because of the high noise level obfuscating the FBF entries too drastically. In Figure 3b, we report the effect of varying ϵ. For each ϵ, we select the T corresponding to the best performance (based on Figure 3a). This shows the expected trend of increasing performance with increasing ϵ, where the DP FlyNN can match the non-DP FlyNN (IS_DP=false) with ϵ close to 1. We present further experimental details and results for different dataset sizes and different FlyNN hyper-parameter configurations in Ram and Sinha (2021a, Appendix D.8).

5 Related work

The k nearest neighbor classification method (kNNC) is a conceptually simple, non-parametric and popular classification method which defers its computational burden to the prediction stage. The consistency properties of kNNC are well studied (Fix and Hodges 1951; Cover and Hart 1967; Devroye et al. 1994; Chaudhuri and Dasgupta 2014). Traditional kNNC assumes training data is stored centrally in a single machine, but such central processing and storing assumptions become unrealistic in the big data era. An effective way to overcome this issue is to distribute the data across multiple machines and use specific distributed computing environment such as Hadoop or Spark with MapReduce paradigm (Anchalia and Roy 2014; Mallio, Triguero, and Herrera 2015; Gonzalez-Lopez, Ventura, and Cano 2018). Zhang et al. (2020) proposed a kNNC algorithm based on the concept of distributed storage and computing for processing large datasets in cyber-physical systems where k-nearest neighbor search is performed locally using a kd-tree. Qiao, Duan, and Cheng (2019) analyzed a distributed kNNC in which data are divided into multiple smaller subsamples, kNNC predictions are made locally in each subsamples and these local predictions are combined via majority voting to make the final prediction. Securely computing kNNC is another closely related field when data is stored in different local devices. Majority of the frameworks that ensure privacy for kNNC often use some sort of secure multi-party computation (SMC) protocols (Zhan, Chang, and Matwin 2008; Xiong, Chitti, and Liu 2006; Qi and Atallah 2008; Schoppmann et al. 2020; Shaul, Feldman, and Rus 2020; Chen et al. 2020).

The federated learning framework involves training statistical models over remote devices while keeping data localized. Such a framework has recently received significant attention with the growth of the storage and computational capabilities of the remote devices within distributed networks especially because learning in such setting differs significantly from tradition distributed environment requiring advances in areas such as security and privacy, large-scale machine learning and distributed optimization. Excellent survey and research questions on this new field can be found in (Li et al. 2020; Kairouz and McMahan 2021). In federated learning, the parameters of a global model is learned in rounds, where in each round a central server sends the current state of the global algorithm (model parameters) to all the parties, each party makes local updates and sends the updates back to the central server (McMahan et al. 2017).

Current distributed kNNC schemes do not directly translate to the federated learning setting since the test point needs to be transmitted to all parties. In most secure kNNC settings considered in the literature, the goal is to keep the training data secure from the party making the test query (Qi and Atallah 2008; Shaul, Feldman, and Rus 2018; Wu et al. 2019) and it is not clear how those approaches extend to the multi-party federated setting where the per-party data (train or test) should remain localized. Of particular relevance is Schoppmann et al. (2020)[3] which proposed a scheme to compute a secure inner-product between any test point and all training points (distributed across parties) and then perform a secure top-k protocol to perform kNNC. This procedure explicitly computes the neighbors for a test point, which involves n secure similarity computations for each test point (on top of the secure top-k protocol). Both these steps require significant communication at inference time. In contrast, our proposed scheme do not require any explicit computation of the nearest-neighbors and hence requires no top-k selection (secure or otherwise). In fact the inference requires *no communication* and the training can be made DP. Moreover, this paper focuses on document classification and leverages the significantly sparse feature representations of training examples. The high sparsity allows the use of *correlated permutations* to compute inner-products. However, the critical use of correlated permutations of the non-sparse indices augmented with padding does not translate to general dense data – in the absence of sparsity, the required correlated permutations would be very large and require multiple rounds of computation for a single similarity computation. Hence it is not clear if these techniques translate to the general kNNC.

6 Limitations & Future Work

A high-level limitation of our work is that we are not presenting a state-of-the-art result, but rather demonstrating how naturally occurring algorithms (FlyHash and FBF) can expand the scope of a canonical ML scheme (NNC) to more learning environments (FL). Our motivations are not to achieve state-of-the-art, but rather to explore and understand the novel unique capabilities of this neurobiological motif.

Another limitation is that the current theoretical connection between FlyNN and kNNC requires assumptions on the class margins and on the distribution of the data (the test point is from a permutation invariant distribution). This limits the scope of the theoretical result though we try to verify the theory with a large number of synthetic and real datasets. We plan to remove such assumptions in our future work.

Acknowledgement. KS gratefully acknowledges funding from the "NSF AI Institute for Foundations of Machine Learning (IFML)" (FAIN: 2019844).

[3] A previous version of Schoppmann et al. (2020) was titled "Private Nearest Neighbors Classification in Federated Databases" (https://eprint.iacr.org/eprint-bin/versions.pl?entry=2018/289) but has since been changed as the focus of the paper was shifted.

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 265–283.

Anchalia, P. P.; and Roy, K. 2014. The k-nearest neighbor algorithm using Mapreduce paradigm. In *International Conference on Intelligent Systems, Modeling and Simulation*.

Charikar, M. S. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 380–388.

Chaudhuri, K.; and Dasgupta, S. 2014. Rate of convergence of nearest neighbor classification. In *Advances in Neural Information Processing Systems*.

Chen, H.; Chilotti, I.; Dong, Y.; Poburinnaya, O.; Razenshteyn, I.; and Riazi, M. S. 2020. Scaling up secure approximate k-nearest neighbor search. In *USENIX security symposium*.

Cover, T.; and Hart, P. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13: 21–27.

Curtin, R. R.; Ram, P.; and Gray, A. G. 2013. Fast exact max-kernel search. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, 1–9. SIAM.

Dasgupta, S.; Sheehan, T. C.; Stevens, C. F.; and Navlakha, S. 2018. A neural data structure for novelty detection. *Proceedings of the National Academy of Sciences*, 115(51): 13093–13098.

Dasgupta, S.; Stevens, C. F.; and Navlakha, S. 2017. A neural algorithm for a fundamental computing problem. *Science*, 358(6364): 793–796.

Defazio, A.; Bach, F.; and Lacoste-Julien, S. 2014. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, 1646–1654.

Devroye, L.; Gyorfi, L.; Krzyzak, A.; and Lugosi, G. 1994. On the strong universal consistency of nearest neighbor regression function estimates. *The annals of Statistics*, 22: 1371–1385.

Fan, R.-E.; Chang, K.-W.; Hsieh, C.-J.; Wang, X.-R.; and Lin, C.-J. 2008. LIBLINEAR: A library for large linear classification. *Journal of machine learning research*, 9(Aug): 1871–1874.

Fix, E.; and Hodges, J. L. 1951. Discriminatory analysis-nonparametric discrimination: consistency properties. *Tech. Rep. California Univ. Berkeley*.

Gonzalez-Lopez, J.; Ventura, S.; and Cano, A. 2018. Distributed nearest neighbor classification for large-scale multi-label data on spark. *Future Generation Computer Systems*, 87: 66–82.

Gottlieb, L.-A.; Kontorovich, A.; and Nisnevitch, P. 2014. Near-optimal sample compression for nearest neighbors. In *Advances in Neural Information Processing Systems*.

Guyon, I. 2003. Design of experiments of the NIPS 2003 variable selection benchmark. In *NIPS 2003 workshop on feature extraction and feature selection*.

Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. R. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Kairouz, P.; and McMahan, H. B. 2021. Advances and Open Problems in Federated Learning. *Foundations and TrendsÂ® in Machine Learning*, 14(1): –.

Kavukcuoglu, K.; Sermanet, P.; Boureau, Y.-L.; Gregor, K.; Mathieu, M.; and LeCun, Y. 2010. Learning convolutional feature hierarchies for visual recognition. In *Advances in neural information processing systems*, 1090–1098.

Keivani, O.; Sinha, K.; and Ram, P. 2017. Improved maximum inner product search with better theoretical guarantees. In *2017 International Joint Conference on Neural Networks (IJCNN)*, 2927–2934. IEEE.

Keivani, O.; Sinha, K.; and Ram, P. 2018. Improved maximum inner product search with better theoretical guarantee using randomized partition trees. *Machine Learning*, 107(6): 1069–1094.

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

Koenigstein, N.; Ram, P.; and Shavitt, Y. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, 535–544.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Larochelle, H.; and Hinton, G. E. 2010. Learning to combine foveal glimpses with a third-order Boltzmann machine. In *Advances in neural information processing systems*, 1243–1251.

Lecun, Y. 1995. The MNIST database of handwritten digits.

Lecun, Y.; and Bengio, Y. 1995. *Convolutional Networks for Images, Speech and Time Series*, 255–258. The MIT Press.

Li, T.; Sahu, A. K.; Talwarkar, A.; and Smith, V. 2020. Federated learning: Challenges, methods and future directions. *IEEE Signal Processing Magazine*, 37(3): 50–60.

Liang, Y.; Ryali, C. K.; Hoover, B.; Grinberg, L.; Navlakha, S.; Zaki, M. J.; and Krotov, D. 2021. Can a Fruit Fly Learn Word Embeddings? *arXiv preprint arXiv:2101.06887*.

Mallio, J.; Triguero, I.; and Herrera, F. 2015. A mapreduce-based k-nearest neighbor approach for big data classification. In *IEEE Trustcom/BigDataSE/ISPA*.

McMahan, H. B.; Moore, E.; Ramage, D.; Hampson, S.; and Arcas, B. A. 2017. Communication efficient learning of deep neural networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*.

Mnih, V.; Heess, N.; Graves, A.; et al. 2014. Recurrent models of visual attention. In *Advances in neural information processing systems*, 2204–2212.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct): 2825–2830.

Qi, Y.; and Atallah, M. J. 2008. Efficient privacy-preserving k-nearest neighbor search. In *International Conference on Distributed Computing Systems*.

Qiao, X.; Duan, J.; and Cheng, G. 2019. Rate of convergence of large scale nearest neighbor classification. In *Advances in Neural Information Processing Systems*.

Ram, P.; and Gray, A. G. 2012. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 931–939.

Ram, P.; and Sinha, K. 2021a. Federated Nearest Neighbor Classification with a Colony of Fruit-Flies: With Supplement. *arXiv preprint arXiv:2112.07157*.

Ram, P.; and Sinha, K. 2021b. FlyNN: Fruit-fly Inspired Federated Nearest Neighbor Classification. *International Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with ICML 2021 (FL-ICML'21)*.

Ryali, C. K.; Hopfield, J. J.; Grinberg, L.; and Krotov, D. 2020. Bio-Inspired Hashing for Unsupervised Similarity Search. *arXiv preprint arXiv:2001.04907*.

Schmidt, M.; Le Roux, N.; and Bach, F. 2017. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2): 83–112.

Schoppmann, P.; Vogelsang, L.; Gascón, A.; and Balle, B. 2020. Secure and Scalable Document Similarity on Distributed Databases: Differential Privacy to the Rescue. *Proceedings on Privacy Enhancing Technologies*, 2020(2): 209–229.

Shaul, H.; Feldman, D.; and Rus, D. 2018. Secure k-ish Nearest Neighbors Classifier. *arXiv preprint arXiv:1801.07301*.

Shaul, H.; Feldman, D.; and Rus, D. 2020. Secure k-ish nearest neighbor classifier. *Proceedings on Privacy Enhancing Technologies*, 2020(3): 42–61.

Sinha, K.; and Ram, P. 2021. Fruit-fly Inspired Neighborhood Encoding for Classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 1470–1480.

Van Rijn, J. N.; Bischl, B.; Torgo, L.; Gao, B.; Umaashankar, V.; Fischer, S.; Winter, P.; Wiswedel, B.; Berthold, M. R.; and Vanschoren, J. 2013. OpenML: A collaborative science platform. In *Joint european conference on machine learning and knowledge discovery in databases*, 645–649. Springer.

Wu, W.; Parampalli, U.; Liu, J.; and Xian, M. 2019. Privacy Preserving K-Nearest Neighbor Classification over Encrypted Database in Outsourced Cloud Environments. *World Wide Web*, 22(1): 101–123.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Xiong, L.; Chitti, S.; and Liu, L. 2006. K nearest neighbor classification across multiple private databases. In *International Conference on Information and Knowledge Management*.

Zhan, J. Z.; Chang, L.; and Matwin, S. 2008. Privacy preserving k-nearest neighbor classification. *ACM Transactions on Intelligent Systems and Technology*, 1(1): 46–51.

Zhang, W.; Chen, X.; Liu, Y.; and Xi, Q. 2020. A distributed storage and computation k-nearest neighbor algorithm based cloud-edge computing for cyber physical systems. *IEEE Access*, 8: 50118–50130.