FlyNN: Fruit-fly Inspired Federated Nearest Neighbor Classification

Parikshit Ram [*1] **Kaushik Sinha** [*2 3]

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 "re-programmed" 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 FlyNNFL. Empirically, we demonstrate that (i) FlyNN matches NNC accuracy across 70 OpenML datasets, (ii) FlyNNFL 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 et al., 2012), convolutions (Lecun & Bengio, 1995), dropout (Hinton et al., 2012), attention mechanisms (Larochelle & 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 et al., 2017; Ryali et al., 2020), outlier detection (Dasgupta et al., 2018) and word embeddings (Liang et al., 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 et al., 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$\rightarrow$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$\rightarrow$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

[*]Equal contribution [1]Mathematics of AI, IBM Research, USA [2]Department of Electrical Engineering & Computer Science, Wichita State University, USA [3]Institute for Foundations of Machine Learning. Correspondence to: Parikshit Ram <p.ram@acm.org>, Kaushik Sinha <kaushik.sinha@wichita.edu>.

This work was presented at the International Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with ICML 2021 (FL-ICML'21). This workshop does not have official proceedings and this paper is non-archival.

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.

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 present a way to perform NNC with FlyNN in a distributed data setting under the FL setup with low communication overhead. The key idea is to train a FlyNN separately on each party and then perform a low communication aggregation 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:

- We present the FlyNN classifier utilizing the FBF and FlyHash, and theoretically present precise conditions under with FlyNN matches NNC.
- We present a training algorithm for FlyNN with distributed data in the FL setup, with low communication, without requiring exchange of the original data.
- We empirically compare FlyNN to NNC and other 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 its low communication overhead.

The paper is organized as follows: We detail our proposed 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.

2. FlyNN based Nearest Neighbor Classifier

In the ensuing presentation, we use lowercase letters (x) scalars or 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 any vector $\boldsymbol{x}$, $\boldsymbol{x}[j]$ denotes its j^{th} index. For any positive integer $k \in \mathbb{N}$, $[k] := \{1, \ldots, k\}$. We will use L to denote the total number of classes and $[L]$ to denote the set of all labels.

We start this section by recalling k-nearest neighbor classification (kNNC). Given a dataset of labeled points $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times [L]$ and 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 $S^k(\boldsymbol{x})$ as follow:

$$\begin{aligned} \hat{y} &\leftarrow \operatorname*{argmax}_{y \in [L]} \left| \left\{ (\boldsymbol{x}_i, y_i) \in S^k(\boldsymbol{x}) : y_i = y \right\} \right|, \\ S^k(\boldsymbol{x}) &= \operatorname*{argmax}_{R \subset S:\ |R|=k} \sum_{(\boldsymbol{x}_i, y_i) \in R} s(\boldsymbol{x}, \boldsymbol{x}_i). \end{aligned} \tag{1}$$

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 $S_1 \cup S_2 \cdots \cup S_\tau$.

We leverage the locality sensitive FlyHash (Dasgupta et al., 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,

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

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 is an upward projection or a *lifting* increasing 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 $\boldsymbol{w}^\top \mathbf{h}_{\mathsf{nv}}$.

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

1: **function** Train$\texttt{FlyNN}$ $(S, m, \rho, s, \gamma, R)$
2: Sample $\mathsf{M} \in \{0,1\}^{m \times d}$ with seed R
3: Initialize $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_L \leftarrow \mathbf{1}_m \in (0,1)^m$
4: **for** $(\boldsymbol{x}, y) \in S$ **do**
5: $\mathbf{h} \leftarrow \Gamma_\rho(\mathsf{M}\boldsymbol{x})$
6: $\boldsymbol{w}_y[i] \leftarrow \gamma \cdot \boldsymbol{w}_y[i] \; \forall i \in [m] \colon \mathbf{h}[i] = 1$
7: **end for**
8: **Return:** $(\mathsf{M}, \{\boldsymbol{w}_l, l \in [L]\})$
9: **end function**
10: **function** Infer$\texttt{FlyNN}(\boldsymbol{x}, \mathsf{M}, \rho, \{\boldsymbol{w}_l, l \in [L]\})$
11: $\mathbf{h} \leftarrow \Gamma_\rho(\mathsf{M}\boldsymbol{x})$
12: **Return:** $\operatorname{argmin}_{l \in [L]} \boldsymbol{w}_l^\top \mathbf{h}$
13: **end function**

2.1. FlyNN: Training and Inference

We extend the use of FBF to the classification setting, an instance of supervised learning. Specifically, we use the high dimensional FBF encoding 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 mitigating overlap between the FBFs of other classes. 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 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 intensity in the FBF; a small value of γ such as 0 implies that the corresponding element in $\boldsymbol{w}_y$ will decay to 0 with the first point resulting in a binary valued FBF. This whole process ensures that $\boldsymbol{x}$ (and points similar to $\boldsymbol{x}$) are considered "inliers" with respect to $\boldsymbol{w}_y$.

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 time complexity and memory overhead of the subroutines in Algorithm 1 for any specific configuration of its hyper-parameters. Proofs are in Appendix A.

Lemma 1. *Given a training set $S \subset \mathbb{R}^d \times [L]$ with n examples, TrainFlyNN (Alg. 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. *Given a trained FlyNN, the inference for any test point $\boldsymbol{x} \in \mathbb{R}^d$ with InferFlyNN (Alg. 1) takes time $O\left(m \cdot \max\left\{s, \log \rho, (\rho L/m)\right\}\right)$ with a memory overhead of $O(\max\{m, L\})$.*

Next we present learning theoretical 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 minimum novelty score $\arg\min_{l \in [L]} \boldsymbol{w}_l^\top h(\boldsymbol{x})$ to label $\boldsymbol{x}$ is equivalent to labeling $\boldsymbol{x}$ with the class whose distribution is "closest" to $\boldsymbol{x}$. This intuition allows us to identify precise conditions where FlyNN mimics the kNNC. The proof is provided in Appendix D.

We present our analysis for the binary classification setting 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. WLOG, for any test point $\boldsymbol{x}$, assume that 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. Unfortunately, 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 predicting the class label of $\boldsymbol{x}$ to be 0. To avoid such a situation, we assume a margin $\eta > 0$ between the classes (Gottlieb et al., 2014) defined as:

Definition 1: We define *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 $\lceil \frac{k+1}{2} \rceil$ of $\boldsymbol{x}$'s nearest neighbors from S are at a distance at most $\eta/2$ from $\boldsymbol{x}$, then all of those $\lceil \frac{k+1}{2} \rceil$ 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

Algorithm 2 Federated FlyNN training with τ parties $V_t, t \in [\tau]$ each with training set S_t and federated inference with test point $\boldsymbol{x}$ at receiving party $V_{t_{\text{in}}}$.

1: **function** TrainFlyNNFL($\{S_t, t \in [\tau]\}, m, \rho, s, \gamma$)
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: **end for**
6: *// All-reduction over all τ parties*
7: $\hat{\boldsymbol{w}}_l[i] \leftarrow \gamma^{\sum_{t\in[\tau]} \log_\gamma \boldsymbol{w}_l^t[i]} \forall i \in [m], \forall l \in [L]$
8: **Return:** $(\mathsf{M}, \{\hat{\boldsymbol{w}}_l, l \in [L]\})$ on each party $V_t, t \in [\tau]$
9: **end function**

$\boldsymbol{x}$ coming from a ***permutation invariant*** distribution agrees with the prediction of kNNC with high probability. 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 $\left(\lceil \frac{k+1}{2} \rceil\right)^{th}$ nearest neighbor from S measured using ℓ_∞ metric. If $\|\boldsymbol{x} - \boldsymbol{x}_*\|_\infty \leq \min\{\frac{\eta}{2}, O(1/s)\}$ then, $\hat{y}_{\texttt{FlyNN}} = \hat{y}_{k\text{NNC}}$ with probability $\geq 1 - \left(O(\frac{\rho n}{m}) + e^{-O(\rho)}\right)$, where $\hat{y}_{\texttt{FlyNN}}$ and $\hat{y}_{k\text{NNC}}$ are respectively the predictions of FlyNN and kNNC on $\boldsymbol{x}$.*

Remark 1. *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 2. *We established conditions under which 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 error rate of the Bayes optimal error. Therefore, by establishing the connection between FlyNN and kNNC, FlyNN has the same statistical guarantee under the conditions of Theorem 3.*

3. Federated NNC via Distributed FlyNN

For the federated learning setup where the training data S is spread across τ parties $V_1, \ldots, V_\tau$ with each party V_t having its own chunk S_t, we present a distributed FlyNNFL learning in Algorithm 2, highlighting the differences from the original FlyNN learning and inference in Blue text.

In TrainFlyNNFL, we ensure that at the end of the training procedure, all the parties $V_t, t \in [\tau]$ have the complete FlyNN model and are *able to perform no-communication inference on any new test point $\boldsymbol{x}$ independent of the other parties*. 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 TrainFlyNN (Algorithm 1) on its chunk S_t and obtains the per-class FBF $\{\boldsymbol{w}_l^t, l \in [L]\}$ (lines 3-5). Finally, a specialized *all-reduce* aggregates all the per-class FBFs $\{\boldsymbol{w}_l^t, l \in [L]\}$ across all parties $t \in [\tau]$ to obtain the final FBFs $\hat{\boldsymbol{w}}_l, l \in [L]$ on all parties (line 6). The following claim establishes exact parity between the FlyNN learned with distributed data $S_t, t \in [\tau]$ and pooled data $S = \cup_{t\in[\tau]} S_t$:

Theorem 4 (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, the per-party final FlyNN $\{\hat{\boldsymbol{w}}_l, l \in [L]\}$ (Alg. 2, line 7) output by TrainFlyNNFL $(\{S_t, t \in [\tau]\}, m, s, \rho, \gamma)$ with random seed R in Algorithm 2 is equal to the FlyNN $\{\boldsymbol{w}_l, l \in [L]\}$ (Alg. 1, line 8) output by TrainFlyNN (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 FlyNNFL 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 FBFs 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 InferFlyNN subroutine in Algorithm 1. The unique capabilities are enabled by the learning dynamics of the FBF in FlyNN. The following is the computational complexity:

Lemma 5 (FlyNNFL training)**.** *Given the FL setup and FlyNN configuration in Theorem 4 with $|S_t| = n_t$, TrainFlyNNFL (Alg. 2) takes time $O\left(n_t m \cdot \max\{s, \log\rho, (L/n_t)\log\tau\}\right)$ with memory overhead $O(m \cdot \max\{s, L\})$ on party $V_t, t \in [\tau]$, and overall communication overhead of $O(mL\tau)$.*

Beyond the aforementioned unique capabilities, Claim 5 shows that, if the data is distributed evenly across all τ parties (that is $n_t = n/\tau$), FlyNNFL training almost achieves linear scaling with respect to τ unless $(L/n)\tau \log\tau$ dominates $\max\{s, \log\rho\}$. The memory overhead per party is same as FlyNN training. The overall communication overhead is linear in τ, m and L. After training, each party can infer with InferFlyNN (Alg. 1) with complexities given by Lemma 2. All proofs are in Appendix B.

Lower communication training. The above TrainFlyNNFL can be extended to have a lower communication overhead (as low as $O(\tau)$) at training time. However, this requires a more sophisticated inference process than InferFlyNN in Algorithm 1 and incurs a small $O(L\rho\tau)$ communication overhead per inference. Note that $\rho \ll m$ given the high level of sparsity of FlyHash. This extension is presented in detail in Appendix C.

Communication setup. The FlyNNFL algorithms are presented here in a peer-to-peer communication setup. However, they will easily transfer to a centralized setup with a "global aggregator" that all parties communicate to. In

that case, in TrainFlyNNFL, 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.

Data movement. TrainFlyNNFL does not require any original data transfer *while still performing an exact FlyNN training* on the pooled data. We transfer the per-party FBFs to all involved parties (either peer-to-peer or through an aggregator). It is not possible to reconstruct the original $\boldsymbol{x}$ from $\mathbf{h}$ given the randomness, winner-take-all (WTA) operation and binarization in FlyHash. Even if the random matrix M is available (removing randomness), it is not possible to reconstruct $\boldsymbol{x}$ from $\mathbf{h}$ – the loss of information from WTA and binarization is not recoverable; yet this hash is sufficient for classification. Recovering training samples from the per-class FBFs (which are an aggregation of per-sample FlyHash-es) is more challenging. Thus, this process ensures data privacy, with no identifiable information leaving their corresponding party.

4. Empirical Evaluation

In this section, we evaluate the empirical performance of FlyNN. First, we compare the performance of FlyNN to NNC to validate its ability to approximate NNC. Then, we demonstrate the scaling of FlyNNFL training on data distributed among multiple parties.

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

- 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.
- We consider high dimensional vision datasets MNIST (Lecun, 1995), Fashion-MNIST (Xiao et al., 2017) and CIFAR (Krizhevsky 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.

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

- **kNNC:** 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 Appendix E for further details.

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 $m \in [2d, 1024d]$.

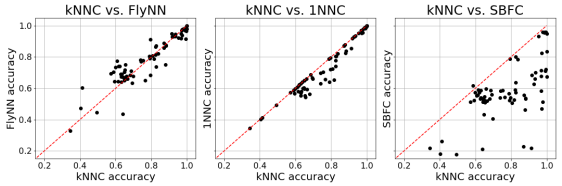

Figure 1: Performance of FlyNN and baselines relative to kNNC on *OpenML* datasets. The scatter plots in Figure 1 compare the best tuned kNNC accuracy against that of FlyNN, 1NNC and SBFC, with a point for each dataset, and the red dashed diagonal marking match to kNNC accuracy.

Table 1: Comparing FlyNN to baselines on OpenML datasets with (i) Fraction of datasets FlyNN exceed baselines, (ii) Number of datasets on which FlyNN has wins/ties/losses over baselines, (iii) Median improvement in normalized accuracy by FlyNN over baseline across all datasets (with p-values for the paired two-sided t-test). • denotes we can reject $\mathbf{H}_0$ at significance level 0.05; ∘ denotes we cannot, (iv) Two-sided Wilcoxon signed rank test p-value.

METHOD	(i) FRAC.	(ii) W/T/L	(iii) IMP. (p-val)	(iv) 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

Evaluation metric to compare across datasets. To compare 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 and a negative value denotes improvement over kNNC. We perform this "normalization" to compare the aggregate performance of different methods across different datasets with distinct best achievable accuracies.

OpenML data. We consider 70 classification (binary and multi-class) datasets from OpenML with d numerical columns and n samples; $d \in [20, 1000], n \in [1000, 20000]$. The results are summarized in Figure 1. In Table 1, the normalized accuracy of all baselines are compared to FlyNN with paired t-tests for the null hypothesis $\mathbf{H}_0$ that FlyNN and the baseline have similar performance at a significance level of 0.05 (over all datasets).

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 over (better FlyNN accuracy)

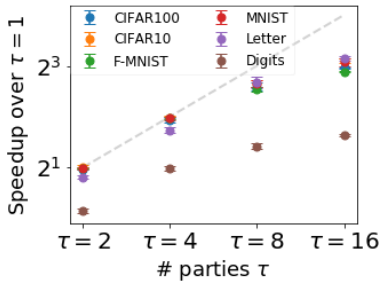

Figure 2: Scaling of FlyNNFL training with τ parties for $\tau = 2, 4, 8, 16$ over single-party training. The gray line marks linear scaling.

and some under (worse accuracy). With 1NNC in the center plot of Figure 1, we see that 1NNC either matches kNNC (since kNNC subsumes 1NNC) or does worse (being under the diagonal). But the right plot for SBFC in Figure 1 indicates that SBFC is quite unable to match kNNC. The results indicate that FlyNN performs very similar to kNNC with some exceptions. We quantify this similarity in Table 1. FlyNN performs comparably to kNNC – null hypothesis $\mathbf{H}_0$ cannot be rejected – while improving the normalized accuracy over 1NNC by a median of around 2.36% across all 70 sets ($\mathbf{H}_0$ can be rejected). 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 (over 25% median improvement), again highlighting the value of high sparsity on real datasets. The p-values are of the similar order of magnitude between the Student t-test and the Wilcoxon signed-rank test.

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

5. Related work

The k nearest neighbor classifier (kNNC) is a conceptually simple, non-parametric classification method whose consistency properties are well studied (Fix & Hodges, 1951; Cover & Hart, 1967; Devroye et al., 1994; Chaudhuri & Dasgupta, 2014) and extensive surveys on kNNC are widely available (Devroye et al., 2013; Chen & Shah, 2018). While the kNNC assumes that the training data is stored and processed in a single machine, such assumption often becomes unrealistic due to distributed nature of the data. An effective way to overcome this issue is to distribute the data across multiple machines and use distributed computing environments such as Hadoop or Spark with MapReduce paradigm (Anchalia & Roy, 2014; Mallio et al., 2015; Gonzalez-Lopez et al., 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 et al. (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. This has been shown to achieve the same rate of convergence as vanilla kNNC up to a multiplicative constant (in terms of regret) that depends on the data dimension. Duan et al. (2020) demonstrated that this multiplicative difference can be eliminated by replacing majority voting with the weighted voting scheme. 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 secure multi-party computation (SMC) protocols (Zhan et al., 2008; Xiong et al., 2006; Qi & Atallah, 2008; Schoppmann et al., 2020; Shaul et al., 2020; Chen et al., 2020).

The federated learning framework involves training statistical models over remote devices while keeping the 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 since learning in such setting differs significantly from the tradition distributed environment. Excellent survey and research questions on this new field can be found in (Li et al., 2020; Kairouz & McMahan, 2021). In federated learning, a global model is learned whose objective function can be represented as any finite sum of objective functions (including neural networks). Learning the parameters of such global model is done in rounds, where in each round a central server sends the current state of the global algorithm (model parameters), each local device makes local updates and sends the updates back to the central server (McMahan et al., 2017). Federated learning has also been used recently to learn collaborative representation of the data stored across local devices for various NLP tasks (Bernal et al., 2021).

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 & Atallah, 2008; Shaul et al., 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.

We rigorously demonstrate both theoretically and empirically how FlyNN is able to match kNNC. Hence our proposed solution presents a way to perform kNNC inference on each party without any communication, and any data transfer during the training is an aggregation of the FlyHash-es of the points. We also outlined a second solution with a lower communication training that only requires to send the FlyHash of the test point.

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

References

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

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

Bernal, D. G., Giaretta, R., Girdzjauskas, S., and Sahlgren, M. Federated word2vec: Leveraging federated learning to encourage collaborative representation learning. *arXiv preprint arXiv:2105.00831*, 2021.

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

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

Chen, G. H. and Shah, D. Explaining the success of nearest neighbor methods in prediction. *Foundations and Trends in Machine Learning*, 10:337–588, 2018.

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

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

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

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

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

Devroye, L., Gyorfi, L., and Lugosi, G. *A probabilistic theory of pattern recognition*. Springer Science & Business Media, 2013.

Duan, J., Qiao, X., and Cheng, G. Statistical guarantees of distributed nearest neighbor classification. In *Advances in Neural Information Processing Systems*, 2020.

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

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

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

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

Kairouz, P. and McMahan, H. B. Advances and open problems in federated learning. *Foundations and TrendsÂ® in Machine Learning*, 14(1):–, 2021. ISSN 1935-8237. doi: 10.1561/2200000083. URL http://dx.doi.org/10.1561/2200000083.

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

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

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

Larochelle, H. and Hinton, G. E. Learning to combine foveal glimpses with a third-order boltzmann machine.

In *Advances in neural information processing systems*, pp. 1243–1251, 2010.

Lecun, Y. The mnist database of handwritten digits, 1995. URL http://yann.lecun.com/exdb/mnist/.

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

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

Liang, Y., Ryali, C. K., Hoover, B., Grinberg, L., Navlakha, S., Zaki, M. J., and Krotov, D. Can a fruit fly learn word embeddings? *arXiv preprint arXiv:2101.06887*, 2021.

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

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

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

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

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

Ryali, C. K., Hopfield, J. J., Grinberg, L., and Krotov, D. Bio-inspired hashing for unsupervised similarity search. *arXiv preprint arXiv:2001.04907*, 2020.

Schoppmann, P., Vogelsang, L., Gascon, A., and Balle, B. Secure and scalable document similarity on distributed databases: Differential privacy to the rescue. *Proceedings on Privacy Enhancing Technologies*, 2020(3):209–229, 2020.

Shaul, H., Feldman, D., and Rus, D. Secure k-ish nearest neighbors classifier. *arXiv preprint arXiv:1801.07301*, 2018.

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

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

Wu, W., Parampalli, U., Liu, J., and Xian, M. Privacy preserving k-nearest neighbor classification over encrypted database in outsourced cloud environments. *World Wide Web*, 22(1):101–123, January 2019. ISSN 1386-145X. doi: 10.1007/s11280-018-0539-4. URL https://doi.org/10.1007/s11280-018-0539-4.

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

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

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

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

A. Proofs for Algorithm 1 complexities

A.1. Proof for Lemma 1

Proof. We can summarize the complexities for the different operations in TrainFlyNN (Algorithm 1) as follows:

- Line 2 takes $O(ms)$ time and memory to generate the random binary lifting matrix M.
- Line 3 takes $O(mL)$ time and memory to initialize the per-class FBFs $\boldsymbol{w}_l, l \in [L]$.
- Each FlyHash in line 5 takes $O(m(s + \log \rho))$ time and $O(m)$ memory.
- FBF $\boldsymbol{w}_y$ update in line 6 takes $O(\rho)$ time since γ is multiplied to only ρ entries in $\boldsymbol{w}_y$ since $\mathbf{h}$ has only ρ non-zero entries.
- Hence the loop 4-7 takes time $O(n(ms + m \log \rho + \rho))$ and maximum $O(m)$ additional memory.
- Given $\rho \ll m$ and $L \ll n$, the total runtime is given by $O(n \cdot m \cdot \max\{s, \log \rho\})$ time and $O(m \cdot \max\{s, L\})$ memory.

This proves the statement of the claim. □

A.2. Proof for Lemma 2

Proof. We can summarize the complexities for the different operations in InferFlyNN (Algorithm 1) as follows:

- The FlyHash operation in line 11 takes time $O(m(s + \log \rho))$ and $O(m)$ memory.
- The operation in line 12 takes time $O(L\rho)$ since each $\boldsymbol{w}_l^\top \mathbf{h}$ takes time $O(\rho)$ since $\mathbf{h}$ only has ρ nonzero entries and $O(L)$ additional memory.
- This leads to an overall runtime of $O(m \cdot \max\{s, \log \rho, (\rho L/m)\})$ and memory overhead of $O(\max\{m, L\})$.

This proves the statement of the claim. □

B. Proofs for Algorithm 2

B.1. Proof of Theorem 4

Proof. Given the per-party data chunk $S_t, t \in [\tau]$, let us consider the FBF $\boldsymbol{w}_l \in (0,1)^m$ for any class $l \in [L]$ learned over the pooled data $S = \cup_{t\in[\tau]} S_t$ using the TrainFlyNN subroutine in Algorithm 1. For any $i \in [m]$ and $l \in [L]$:

$$\begin{aligned}\boldsymbol{w}_l[i] &= 1 \cdot \underbrace{\gamma \cdot \gamma \cdot \cdots \cdot \gamma}_{\#\text{ times } \mathbf{h}[i]=1 \text{ for some } (\boldsymbol{x},y)\in S, y=l} \\ &= \gamma^{|\{(\boldsymbol{x},y)\in S: y=l, \mathbf{h}=\Gamma_\rho(\mathsf{M}\boldsymbol{x}), \mathbf{h}[i]=1\}|}. \end{aligned} \tag{3}$$

By the same argument, the FBF $\boldsymbol{w}_l^t$ learned with data chunk S_t for a class l (Algorithm 2, line 4) can be summarized as follows for any $i \in [m]$:

$$\boldsymbol{w}_l^t[i] = \gamma^{|\{(\boldsymbol{x},y)\in S_t: y=l, \mathbf{h}=\Gamma_\rho(\mathsf{M}\boldsymbol{x}), \mathbf{h}[i]=1\}|}. \tag{4}$$

Then the all-reduced FBF $\hat{\boldsymbol{w}}_l$ for a class l (Algorithm 2, ine 6) is given by the following for any $i \in [m]$:

$$\hat{\boldsymbol{w}}_l[i] = \gamma^{\sum_{t\in[\tau]} \log_\gamma \boldsymbol{w}_l^t[i]} \tag{5}$$

$$\overset{(A)}{=} \gamma^{\sum_{t\in[\tau]} |\{(\boldsymbol{x},y)\in S_t: y=l, \mathbf{h}=\Gamma_\rho(\mathsf{M}\boldsymbol{x}), \mathbf{h}[i]=1\}|} \tag{6}$$

$$\overset{(B)}{=} \gamma^{\left|\cup_{t\in[\tau]} \{(\boldsymbol{x},y)\in S_t: y=l, \mathbf{h}=\Gamma_\rho(\mathsf{M}\boldsymbol{x}), \mathbf{h}[i]=1\}\right|} \tag{7}$$

$$\overset{(C)}{=} \gamma^{\left|\left\{(\boldsymbol{x},y)\in \underbrace{\cup_{t\in[\tau]} S_t}_{S}: y=l, \mathbf{h}=\Gamma_\rho(\mathsf{M}\boldsymbol{x}), \mathbf{h}[i]=1\right\}\right|} \tag{8}$$

$$= \gamma^{|\{(\boldsymbol{x},y)\in S: y=l, \mathbf{h}=\Gamma_\rho(\mathsf{M}\boldsymbol{x}), \mathbf{h}[i]=1\}|} \tag{9}$$

$$\overset{(D)}{=} \boldsymbol{w}_l[i], \tag{10}$$

where (A) is obtained from (4), (B) is obtained from the fact that the sets $S_t, t \in [\tau]$ are disjoint (no data shared between parties), (C) is from the fact that a union of subsets of disjoint sets (S_t) is the same as a subset of union of disjoint sets $\cup_t S_t$. (D) follows from (3).

Since the above holds for all $i \in [m]$, we can say that $\boldsymbol{w}_l = \hat{\boldsymbol{w}}_l \forall l \in [L]$, proving the statement of the claim. □

B.2. Proof of Lemma 5

Proof. We begin with recalling that the all-reduce operation can be performed efficiently in a peer-to-peer communication setup where the τ parties can be organized as a binary tree of depth $O(\log \tau)$. Then the communication at each level of the tree can be done in parallel for each independent subtree at that level. Consider the object being all-reduced to be of size $O(c)$. Then in the first round of communication, $O(\tau/2)$ pairs of parties combine their objects in parallel in time $O(c)$ with total communication $O(c\tau/2)$ and $O(c)$ memory overhead in each of the parties. In the second round, $O(\tau/4)$ pairs of parties combine their objects in parallel again in time $O(c)$ with total communication $O(c\tau/4)$ with $O(c)$ memory overhead in $O(\tau/2)$ of the parties. Going up the tree to the root then takes time $O(c \log \tau)$. The total communication cost is $O(c\tau)$.

The communication first goes bottom up from the leaves to the root, which then has the final all-reduced result. Then this result is sent to each party top-down from the root (in a corresponding manner) so that eventually all parties have the all-reduced result in time $O(c \log \tau)$ with total $O(c\tau)$ communication.

Based on the above complexities of the all-reduce operation, we can summarize the complexities for the different operations in TrainFlyNNFL (Algorithm 2) as follows:

- The broadcast of the random seed in line 2 can be done with an all-reduce in $O(\log \tau)$ time and $O(\tau)$ total communication and $O(1)$ memory overhead in each party.
- On party V_t, the invocation of TrainFlyNN in line 4 on data chunk S_t of size n_t takes $O(n_t \cdot m \cdot \max\{s, \log \rho\})$ and $O(m \cdot \max\{s, L\})$ memory from Lemma 1 and no communication cost.
- The all-reduce of the per-party per-class FBFs in line 6 takes time $O(m \cdot L \cdot \log \tau)$ with $O(m \cdot L \cdot \tau)$ total communication, and $O(m \cdot L)$ memory overhead per-party.

Putting them all together gives us the per-party time complexity of $O(n_t \cdot m \cdot \max\{s, \log \rho, (L/n_t) \log \tau\})$, memory overhead of $O(m \cdot \max\{s, L\})$ and total communication among all parties of $O(m \cdot L \cdot \tau)$, giving us the statement of the claim. □

C. Lower communication FlyNNFL

In many situations, an all-reduce operation is not desirable,

Algorithm 3 Federated FlyNN training with τ parties $V_t, t \in [\tau]$ each with training set S_t and federated inference with test point $\boldsymbol{x}$ at receiving party $V_{t_{\text{in}}}$.

1: **function** LCTrainFlyNNFL($\{S_t, t \in [\tau]\}, m, \rho, s, \gamma$)
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: **end for**
6: **Return:** $(\mathsf{M}, \{\boldsymbol{w}_l^t, l \in [L]\})$ on each party $V_t, t \in [\tau]$
7: **end function**
8: **function** LCInferFlyNNFL($\boldsymbol{x}, \mathsf{M}, \rho, \{\boldsymbol{w}_l^t, l \in [L], t \in [\tau]\}$)
9: Generate $\mathbf{h} \leftarrow \Gamma_\rho(\mathsf{M}x)$ on receiving party $V_{t_{\text{in}}}$
10: Broadcast $\mathbf{h}$ to all parties $V_t, t \neq t_{\text{in}} \in [\tau]$
11: **for** each party $V_t, t \in [\tau]$ **do**
12: $g_l^t \leftarrow \{\}$ (empty dictionary)
13: $g_l^t[i] \leftarrow \boldsymbol{w}_l^t[i]\ \forall i \in [m] : \mathbf{h}[i] = 1, \forall l \in [L]$
14: **end for**
15: Gather $\{g_l^t \forall l \in [L], t \in [\tau]\}$ on $V_{t_{\text{in}}}$
16: $g_l \leftarrow \{\}, l \in [L]$ (empty dictionary)
17: $g_l[i] \leftarrow \gamma^{\sum_{t\in[\tau]} \log_\gamma g_l^t[i]}, \forall i \in [m] : \mathbf{h}[i] = 1, \forall l \in [L]$
18: **Return:** $\arg\min_{l\in[L]} \sum_{i:\mathbf{h}[i]=1} g_l[i]$
19: **end function**

and lower communication at training is preferred. Hence, we propose a second FlyNNFL learning scheme, detailed in LCTrainFlyNNFL (Alg. 3). The algorithm starts like TrainFlyNNFL by generating and broadcasting a random seed R to all parties $t \in [\tau]$ (line 2), each of which then invoke TrainFlyNN (Alg. 1) on their respective chunks S_t to obtain the per-class FBFs $\{\boldsymbol{w}_l^t, l \in [L]\}$ (lines 3-5). The difference is the absence of the final all-reduce step; instead, all parties retain their own distinct set of per-class FBFs (line 6). This setup requires a new inference procedure, detailed in LCInferFlyNNFL (Alg. 2) which requires all parties to participate. The party $V_{t_{\text{in}}}$ receiving the test point $\boldsymbol{x}$ generates its FlyHash $\mathbf{h}$ (line 9). Then, the indices $i_{\mathbf{h}}$ of the nonzero entries in $\mathbf{h}$ are broadcast to all parties $V_t, t \neq t_{\text{in}} \in [\tau]$ (line 10). Each party V_t collects the j^{th} elements $(j \in i_{\mathbf{h}})$ of their per-class FBFs $\{\boldsymbol{w}_l^t, l \in [L]\}$ into a dictionary g_l^t (line 11-14). Party $V_{t_{\text{in}}}$ *gathers* $g_l^t, l \in [L]$ from all parties $V_t, t \in [\tau]$ into g_l (line 15-17), and generates the per-class scores by summing all entries in g_l, predicting the class with the lowest score (line 18). We show that the predictions made by LCInferFlyNNFL are same as the predictions made by InferFlyNN (Alg. 1) if we trained with the pooled data $S = \cup_{t=1}^{\tau} S_t$ using TrainFlyNN (Alg. 1):

Theorem 6 (Federated inference parity)**.** *Given the FL setup and FlyNN configuration in Theorem 4 trained with LCTrainFlyNNFL $(\{S_t, t \in [\tau]\}, m, s, \rho, c)$ in Alg. 2, and any test point $\boldsymbol{x} \in \mathbb{R}^d$ at party $V_{t_{\text{in}}}$, let $\hat{y}_{\mathsf{FL}}(\boldsymbol{x})$ be the output of LCInferFlyNNFL $(\boldsymbol{x}, \mathsf{M}, \rho, \{\boldsymbol{w}_l^t, l \in [L], t \in [\tau]\})$. Let $\hat{y}(\boldsymbol{x})$ be the output of InferFlyNN $(\boldsymbol{x}, \mathsf{M}, \rho, \{\boldsymbol{w}_l, l \in [L]\})$ (Alg. 1) where the FlyNN $\{\boldsymbol{w}_l, l \in [L]\}$ is trained with TrainFlyNN (S, m, s, ρ, c) (Alg. 1) with the pooled set $S = \cup_{t\in[\tau]} S_t$. Then, $\hat{y}_{\mathsf{FL}}(\boldsymbol{x}) = \hat{y}(\boldsymbol{x})$.*

Proof. Given a test point $\boldsymbol{x}$ and corresponding FlyHash $\mathbf{h} = \Gamma_\rho(\mathsf{M}\boldsymbol{x})$, and the per-party data chunk $S_t, t \in [\tau]$, let us consider its score $\boldsymbol{w}_l^\top \mathbf{h}$ for any class $l \in [L]$ generated during the inference with InferFlyNN subroutine in Algorithm 1 using the FBF $\boldsymbol{w}_l \in (0,1)^m$ learned over the pooled data $S = \cup_{t\in[\tau]} S_t$ with the TrainFlyNN subroutine in Algorithm 1. For any $l \in [L]$:

$$\boldsymbol{w}_l^\top \mathbf{h} = \sum_{i\in[m]:\mathbf{h}[i]=1} \boldsymbol{w}_l[i] = \sum_{i\in[m]:\mathbf{h}[i]=1} \gamma^{\left|\left\{(\boldsymbol{x}',y')\in S : y'=l, \mathbf{h}'=\Gamma_\rho(\mathsf{M}\boldsymbol{x}'), \mathbf{h}'[i]=1\right\}\right|}, \tag{11}$$

where the last equality is obtained from the definition of $\boldsymbol{w}_l[i]$ in (3). Given $\boldsymbol{w}_l^\top \mathbf{h}$ for each $l \in [L]$, the predicted label generated by InferFlyNN is given by $\hat{y}(\boldsymbol{x}) = \arg\min_{l\in[L]} \boldsymbol{w}_l^\top \mathbf{h}$.

In the LCInferFlyNNFL subroutine in Algorithm 2, in line 21, on party V_t, the dictionary elements in g_l^t corresponding to $i \in [m]$ such that $\mathbf{h}[i] = 1$ are set as:

$$g_l^t[i] = \boldsymbol{w}_l^t[i] = \gamma^{\left|\left\{(\boldsymbol{x}',y')\in S_t : y'=l, \mathbf{h}'=\Gamma_\rho(M\boldsymbol{x}'), \mathbf{h}'[i]=1\right\}\right|}, \tag{12}$$

where the last equality follows from $\boldsymbol{w}_l^t[i]$ defined in (4). Then the aggregate entries of the aggregated dictionary g_l on $V_{t_{\text{in}}}$ for $i \in [m]$ such that $\mathbf{h}[i] = 1$ in line 25 are given as:

$$g_l[i] = \gamma^{\sum_{t\in[\tau]} \log_\gamma g_l^t[i]} \tag{13}$$

$$\overset{(A)}{=} \gamma^{\sum_{t\in[\tau]} \left|\left\{(\boldsymbol{x}',y')\in S_t : y'=l, \mathbf{h}'=\Gamma_\rho(\mathsf{M}\boldsymbol{x}'), \mathbf{h}'[i]=1\right\}\right|} \tag{14}$$

$$\overset{(B)}{=} \gamma^{\left|\cup_{t\in[\tau]}\left\{(\boldsymbol{x}',y')\in S_t : y'=l, \mathbf{h}'=\Gamma_\rho(\mathsf{M}\boldsymbol{x}'), \mathbf{h}'[i]=1\right\}\right|} \tag{15}$$

$$\overset{(C)}{=} \gamma^{\left|\left\{(\boldsymbol{x}',y')\in \underbrace{\cup_{t\in[\tau]} S_t}_{S} : y'=l, \mathbf{h}'=\Gamma_\rho(\mathsf{M}\boldsymbol{x}'), \mathbf{h}'[i]=1\right\}\right|} \tag{16}$$

$$= \gamma^{\left|\left\{(\boldsymbol{x}',y')\in S : y'=l, \mathbf{h}'=\Gamma_\rho(\mathsf{M}\boldsymbol{x}'), \mathbf{h}'[i]=1\right\}\right|} \tag{17}$$

where (A) is obtained from (12), (B) is obtained from the fact that the sets $S_t, t \in [\tau]$ are disjoint (no data shared between parties), (C) is from the fact that a union of subsets of disjoint sets (S_t) is the same as a subset of union of disjoint sets $\cup_t S_t$. Then we can say that:

$$\sum_{i\in[m]:\mathbf{h}[i]=1} g_l[i] = \sum_{i\in[m]:\mathbf{h}[i]=1} \gamma^{\left|\left\{(\boldsymbol{x}',y')\in S: y'=l, \mathbf{h}'=\Gamma_\rho(\mathsf{M}\boldsymbol{x}'), \mathbf{h}'[i]=1\right\}\right|} = \boldsymbol{w}_l^\top \mathbf{h}.$$

The last equality holds from (11). Since the above holds for all $l \in [L]$, and the predicted label generated by LCInferFlyNNFL is given by $\hat{y}_{\mathsf{FL}}(\boldsymbol{x}) = \arg\min_{l\in[L]} \sum_{i\in[m]:\mathbf{h}[i]=1} g_l[i]$ we have:

$$\begin{aligned}\hat{y}_{\mathsf{FL}}(\boldsymbol{x}) &= \arg\min_{l\in[L]} \sum_{i:\mathbf{h}[i]=1} g_l[i] \\ &= \arg\min_{l\in[L]} \boldsymbol{w}_l^\top \mathbf{h} \\ &= \hat{y}(\boldsymbol{x}),\end{aligned}$$

giving us the statement of the claim. □

The following results show how this version of FlyNNFL training reduces the communication overhead without increasing training time, but transfers some communication overhead to the inference:

Lemma 7 (LCFlyNNFL training)**.** *Given the FL setup and FlyNN configuration in Theorem 4 with $|S_t| = n_t$, LCTrainFlyNNFL (Alg. 2) takes time $O\left(n_t m \cdot \max\{s, \log\rho\}\right)$ with a memory overhead of $O(m \cdot \max\{s, L\})$ per party $V_t, t \in [\tau]$, and a communication overhead of $O(\tau)$.*

Proof. Based on the complexities of the all-reduce operation described in §B.2 in the proof for Lemma 5, we can summarize the complexities for the different operations in LCTrainFlyNNFL (Algorithm 2) as follows:

- The broadcast of the random seed in line 10 can be done with an all-reduce in $O(\log\tau)$ time and $O(\tau)$ total communication and $O(1)$ memory overhead in each party.
- On party V_t, the invocation of TrainFlyNN in line 12 on data chunk S_t of size n_t takes $O(n_t \cdot m \cdot \max\{s, \log\rho\})$ and $O(m \cdot \max\{s, L\})$ memory from Lemma 1 and no communication cost.

Putting them all together gives us the per-party time complexity of $O(n_t \cdot m \cdot \max\{s, \log\rho\})$, memory overhead of $O(m \cdot \max\{s, L\})$ and total communication among all parties of $O(\tau)$, giving us the statement of the claim. □

Lemma 8 (LCFlyNNFL inference)**.** *Given the partial FlyNN on each party $V_t, t \in [\tau]$, the inference for any test point $\boldsymbol{x} \in \mathbb{R}^d$ using LCInferFlyNNFL (Alg. 2) takes time $O\left(\max\{m \cdot \max\{s, \log\rho\}, L\rho\tau\}\right)$, with a memory overhead of $O(\max\{m, \rho L\})$ on the receiving party $V_{t_{\mathsf{in}}}$ and $O(\rho L)$ on the remaining parties, and a communication overhead of $O(\tau\rho L)$.*

Proof. We can summarize the complexities for the different operations in LCInferFlyNNFL (Algorithm 2) as follows:

- The FlyHash operation in line 17 takes time $O(m(s + \log\rho))$ and $O(m)$ memory on party $V_{t_{\mathsf{in}}}$.
- The broadcast of $\mathbf{h}$ to all parties $V_t, t \in [\tau], t \neq t_{\mathsf{in}}$ in line 18 takes time $O(\rho \cdot \log\tau)$, $O(\rho)$ additional memory overhead on each party, and $O(\rho \cdot \tau)$ total communication. This is because $\mathbf{h}$ has only ρ nonzero entries, so we need to just send the indices of the nonzero entries in $\mathbf{h}$.
- The per-party creation of the dictionary g_l^t on party V_t in line 20-21 takes time $O(L \cdot \rho)$ and $O(L \cdot \rho)$ memory overhead in each party.
- The gather operation in line 23 and the computation of the aggregated dictionary g_l on $V_{t_{\mathsf{in}}}$ in lines 24-25 can be done with a specialized all-reduce (as described in §B.2) in time $O(L \cdot \rho \cdot \log\tau)$, with additional $O(L \cdot \rho)$ memory on each party, and $O(L \cdot \rho \cdot \tau)$ overall communication cost.
- The computation of the per-class novelty scores and computation of the $\arg\min_l$ in line 26 takes time $O(L \cdot \rho)$ and $O(L)$ additional memory on $V_{t_{\mathsf{in}}}$.

Putting it all together, we see that,

- On $V_{t_{\mathsf{in}}}$, the time complexity is $O(\max\{m \cdot \max\{s, \log\rho\}, L \cdot \rho \cdot \log\tau\})$ and the memory overhead of $O(\max\{m, L \cdot \rho\})$.
- On each $V_t, t \in [\tau], t \neq t_{\mathsf{in}}$, the time complexity of $O(L \cdot \rho)$ and memory overhead of $O(L \cdot \rho)$.
- The overall communication overhead is $O(L \cdot \rho \cdot \tau)$.

This gives us the statement of the claim. □

Lemma 7 shows that the FlyNN training time **and** communication overhead is improved over the previous FlyNNFL training (Lemma 5), with the communication overhead reduced from $O(\tau m L)$ to just $O(\tau)$. However, Lemma 8 shows that each inference incurs a communication overhead of $O(\tau\rho L)$; note that $\rho \ll m$ and hence $O(\tau\rho L)$ is much better than $O(\tau m L)$ communication in Lemma 5. The inference time (compared to Claim 2) includes $L\rho\tau$ along with $m \cdot \max\{s, \log\rho\}$.

D. Proof of Algorithm 1 learning theoretic properties

D.1. Preliminaries & notations

We denote a single row of a lifting matrix M by $\theta \in \{0,1\}^d$ drawn i.i.d. from Q, the uniform distribution over all vectors

in $\{0,1\}^d$ with exactly s ones, satisfying $s \ll d$. We use an alternate formulation of the winner-take-all strategy as suggested in Dasgupta et al. (2018), where for any $\boldsymbol{x} \in \mathbb{R}^d, \tau_{\boldsymbol{x}}$ is a threshold that sets largest ρ entries of $\mathsf{M}\boldsymbol{x}$ to one (and the rest to zero) in expectation. Specifically, for a given $\boldsymbol{x} \in \mathbb{R}^d$ and for any fraction $0 < f < 1$, we define $\tau_{\boldsymbol{x}}(f)$ to be the top f-fractile value of the distribution $\theta^\top \boldsymbol{x}$, where $\theta \sim Q$:

$$\tau_{\boldsymbol{x}}(f) = \sup\{v : \Pr_{\theta\sim Q}(\theta^\top \boldsymbol{x} \geq v) \geq f\} \tag{18}$$

We note that for any $0 < f < 1$, $\Pr_{\theta\sim Q}(\theta^\top \boldsymbol{x} \geq \tau_{\boldsymbol{x}}(f)) \approx f$, where the approximation arises from possible discretization issues. For convenience, henceforth we will assume that this is an equality:

$$\Pr_{\theta\sim Q}(\theta^\top \boldsymbol{x} \geq \tau_{\boldsymbol{x}}(f)) = f \tag{19}$$

For any two $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$, we define $q(\boldsymbol{x}, \boldsymbol{x}') = \Pr_{\theta\sim Q}\left(\theta^\top \boldsymbol{x}' \geq \tau_{\boldsymbol{x}'}(\rho/m) \mid \theta^\top \boldsymbol{x} \geq \tau_{\boldsymbol{x}}(\rho/m)\right)$. This can be interpreted as follows – with $\mathbf{h} = h(\boldsymbol{x}), \mathbf{h}' = h(\boldsymbol{x}')$ as the FlyHashes of $\boldsymbol{x}$ and $\boldsymbol{x}'$, respectively, $q(\boldsymbol{x}, \boldsymbol{x}')$ is the probability that $\mathbf{h}'[j] = 1$ given that $\mathbf{h}[j] = 1$, for any specific $j \in [m]$.

We begin by analyzing the binary classification performance of FlyNN trained on a training set $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n_0+n_1} \subset \mathbb{R} \times \{0,1\}$, where $S = S^1 \cup S^0$, S^0 is a subset of S having label 0, and S^1 is a subset of S having label 1, satisfying $|S^0| = n_0$, $|S^1| = n_1$ and $n = n_0 + n_1$. For appropriate choice of m, let $\boldsymbol{w}_0, \boldsymbol{w}_1 \in \{0,1\}^m$ be the FBFs constructed using S^0 and S_1 respectively.

D.2. Connection to kNNC

We first present the following lemmas which will be required to prove Theorem 3.

Lemma 9 (Expected novelty response (Dasgupta et al., 2018))**.** *Suppose that inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ are first presented with $\boldsymbol{x}_i \to \boldsymbol{y}_i \to \mathsf{h}_i$, where $\boldsymbol{y}_i = \mathsf{M}\boldsymbol{x}_i, \mathsf{h}_i = h(\boldsymbol{x}_i)$, and $\boldsymbol{w}$ is the FBF constructed using $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. Then a subsequent input $\boldsymbol{x}$ is presented with $\boldsymbol{x} \to \boldsymbol{y} \to \mathsf{h}$, where $\boldsymbol{y} = \mathsf{M}\boldsymbol{x}, \mathsf{h} = h(\boldsymbol{x})$.*

(a) The m random vectors $(\boldsymbol{y}_1[j], \ldots, \boldsymbol{y}_n[j], \mathsf{h}_1[j], \ldots, \mathsf{h}_n[j], \boldsymbol{y}[j], \mathsf{h}[j], \boldsymbol{w}[j]), 1 \leq j \leq m$, (over the random choice choice of M) are independent and identically distributed.

(b) The novelty response to $\boldsymbol{x}$ has expected value

$$\mu = \mathbb{E}(\boldsymbol{w}^\top h(\boldsymbol{x})/\rho) = \Pr_{\theta\sim Q}\left(\theta^\top \boldsymbol{x}_1 < \tau_{\boldsymbol{x}_1}, \ldots, \theta^\top \boldsymbol{x}_n < \tau_{\boldsymbol{x}_n}, \theta^\top \boldsymbol{x} \geq \tau_{\boldsymbol{x}}\right)$$

Lemma 10 (Bounds on expected novelty response (Dasgupta et al., 2018))**.** *The expected value μ from Lemma 9 can be bounded as follows:*

(a) Lower bound: $\mu \geq 1 - \sum_{i=1}^n q(\boldsymbol{x}, \boldsymbol{x}_i)$.

(b) Upper bound: for any $1 \leq l \leq n$, $\mu \leq 1 - q(\boldsymbol{x}, \boldsymbol{x}_l)$.

Lemma 11 ((Dasgupta et al., 2018))**.** *Pick any $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$. Suppose that for all $i \in [d]$, $\boldsymbol{x}'[i] \geq \boldsymbol{x}[i] - \Delta/s$, where $\Delta = \frac{1}{2}\left(\tau_{\boldsymbol{x}}(\rho/2m) - \tau_{\boldsymbol{x}}(\rho/m)\right)$. then $q(\boldsymbol{x}, \boldsymbol{x}') \geq 1/2$.*

Corollary 12 ((Dasgupta et al., 2018))**.** *Fix any $\boldsymbol{x}' \in \mathbb{R}^d$ and pick $\boldsymbol{x}$ from any permutation invariant distribution over $\mathbb{R}^d$. then the expected value of $q(\boldsymbol{x}, \boldsymbol{x}')$, over the choice of $\boldsymbol{x}$ is ρ/m.*

Lemma 13. *Fix any $\boldsymbol{x} \in \mathbb{R}^d$ and let $h(\boldsymbol{x}) \in \{0,1\}^m$ be its FlyHash using equation 2. For any integer k, let $\boldsymbol{x}_*^i$ be the $\left(\lceil \frac{k+1}{2} \rceil\right)^{th}$ nearest neighbor of $\boldsymbol{x}$ in S^i measured using ℓ_∞ metric. Let $A_{S^1} = \{\theta : \cap_{(\boldsymbol{x}', y') \in S^1} \theta^\top \boldsymbol{x}' < \tau_{\boldsymbol{x}'}(\rho/m)\}$ and $A_{S^0} = \{\theta : \cap_{(\boldsymbol{x}', y') \in S^0} \theta^\top \boldsymbol{x}' < \tau_{\boldsymbol{x}'}(\rho/m)\}$. Then the following holds, where the expectation is taken over the random choice of projection matrix M.*

(i) $\mathbb{E}_{\mathsf{M}}(\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho}) = \Pr_{\theta\sim Q}\left(A_{S^1} \mid \theta^\top \boldsymbol{x} \geq \tau_{\boldsymbol{x}}(\rho/m)\right)$

(ii) $\mathbb{E}_{\mathsf{M}}(\frac{\boldsymbol{w}_0^\top h(\boldsymbol{x})}{\rho}) = \Pr_{\theta\sim Q}\left(A_{S^0} \mid \theta^\top \boldsymbol{x} \geq \tau_{\boldsymbol{x}}(\rho/m)\right)$

(iii) $\mathbb{E}_{\mathsf{M}}(\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho}) \geq 1 - \sum_{\boldsymbol{x}' \in S^1} q(\boldsymbol{x}, \boldsymbol{x}')$

(iv) $\mathbb{E}_{\mathsf{M}}(\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho}) \leq 1 - q(\boldsymbol{x}, \boldsymbol{x}_*^1)$

(v) $\mathbb{E}_{\mathsf{M}}(\frac{\boldsymbol{w}_0^\top h(\boldsymbol{x})}{\rho}) \geq 1 - \sum_{\boldsymbol{x}' \in S^0} q(\boldsymbol{x}, \boldsymbol{x}')$

(vi) $\mathbb{E}_{\mathsf{M}}(\frac{\boldsymbol{w}_0^\top h(\boldsymbol{x})}{\rho}) \leq 1 - q(\boldsymbol{x}, \boldsymbol{x}_*^0)$

Proof. Part (i) and (ii) follows from simple application of Lemma 9 to class specific FBFs. Part (iii) and (v) follows from simple application of Lemma 10 to class specific FBFs. For part (iv), simple application of Lemma 10 to FBF $\boldsymbol{w}_1$ ensures that for any $\boldsymbol{x}' \in S^1, \mathbb{E}_{\mathsf{M}}(\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho}) \leq 1 - q(\boldsymbol{x}, \boldsymbol{x}')$. Clearly, $\mathbb{E}_{\mathsf{M}}(\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho}) \leq 1 - q(\boldsymbol{x}, \boldsymbol{x}_*^1)$. Applying similar argument, part (vi) also holds. □

Lemma 14. *Let $\boldsymbol{w}_0$ and $\boldsymbol{w}_1$ be the FBFs constructed using S^0 and S^1. For any $\boldsymbol{x} \in \mathbb{R}^d$ let $\mu_0 = \mathbb{E}_{\mathsf{M}}\left(\frac{\boldsymbol{w}_0^\top h(\boldsymbol{x})}{\rho}\right)$ and $\mu_1 = \mathbb{E}_{\mathsf{M}}\left(\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho}\right)$ Then, for any $\epsilon > 0$ the following holds,*

(i) $\Pr\left(\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho} \geq (1+\epsilon)\mu_1\right) \leq \exp(-\epsilon^2 \rho \mu_1/3)$

(ii) $\Pr\left(\frac{\boldsymbol{w}_0^\top h(\boldsymbol{x})}{\rho} \geq (1+\epsilon)\mu_0\right) \leq \exp(-\epsilon^2 \rho \mu_0/3)$

(iii) $\Pr\left(\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho} \leq (1-\epsilon)\mu_1\right) \leq \exp(-\epsilon^2 \rho \mu_1/2)$

(iii) $\Pr\left(\frac{\boldsymbol{w}_0^\top h(\boldsymbol{x})}{\rho} \leq (1-\epsilon)\mu_0\right) \leq \exp(-\epsilon^2 \rho \mu_0/2)$

Proof. We will only prove part (i) since part (ii) is similar. Let $\mathbf{h} = h(\boldsymbol{x}), \mathbf{h}_1 = h(\boldsymbol{x}_1), \ldots, \mathbf{h}_{n_1} = h(\boldsymbol{x}_{n_1})$ be the FlyHashes of $\boldsymbol{x}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_1}$ that belongs to S^1. Define random variables $U_1, \ldots, U_m \in \{0,1\}$ as follows:

$$U_j = \begin{cases} 1, & \text{if } \mathbf{h}_1[j] = \cdots = \mathbf{h}_{n_1}[j] = 0 \text{ and } \mathbf{h}[j] = 1 \\ 0, & \text{otherwise} \end{cases}$$

The U_j are i.i.d. and

$$\begin{aligned}\mathbb{E}_{\mathsf{M}}(U_j) &= \Pr_{\mathsf{M}}(\mathbf{h}[j]=1)\times \\ &\quad \Pr_{\mathsf{M}}\left(\mathbf{h}_1[j]=\cdots=\mathbf{h}_{n_1}[j]=0 \mid \mathbf{h}[j]=1\right) \\ &= \frac{\rho}{m}\mathbb{E}_{\mathsf{M}}\left(\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho}\right)\end{aligned}$$

where we have used the fact that $\Pr_{\mathsf{M}}(h(\boldsymbol{x})_j = 1) = \Pr_{\theta\sim Q}(\theta^\top \boldsymbol{x} \geq \tau_{\boldsymbol{x}}(\rho/m)) = \rho/m$ and using Lemma 2 of the supplementary material of (Dasgupta et al., 2018), $\Pr_{\mathsf{M}}\left(\mathbf{h}_1[j]=\cdots=\mathbf{h}_{n_1}[j]=0 \mid \mathbf{h}[j]=1\right) = \mathbb{E}_{\mathsf{M}}\left(\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho}\right)$. Therefore, $\mathbb{E}_{\mathsf{M}}(U_1+\cdots+U_m) = \rho \cdot \mathbb{E}_{\mathsf{M}}\left(\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho}\right)$. Let $\mu_1 = \mathbb{E}_{\mathsf{M}}\left(\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho}\right)$. By multiplicative Chernoff bound for any $0<\epsilon<1$, we have,

$$\begin{aligned}\Pr_M\left(U_1+\cdots+U_m \geq (1+\epsilon)\rho\mu_1\right) &\leq \exp(-\epsilon^2\rho\mu_1/3) \\ \Pr_M\left(U_1+\cdots+U_m \leq (1-\epsilon)\rho\mu_1\right) &\leq \exp(-\epsilon^2\rho\mu_1/2)\end{aligned}$$

Noticing that $U_1+\cdots+U_m = \boldsymbol{w}_1^\top h(\boldsymbol{x})$, the result follows. □

D.3. Proof of Theorem 3

Proof. Without loss of generality, assume that kNNC prediction of $\boldsymbol{x}$ is 1. For the case when kNNC prediction is 0 is similar. Prediction of FlyNN on $\boldsymbol{x}$ agrees with the prediction of kNNC whenever $\left(\boldsymbol{w}_1^\top h(\boldsymbol{x})/\rho\right) < \left(\boldsymbol{w}_0^\top h(\boldsymbol{x})/\rho\right)$. We first show that $\mathbb{E}_{\mathsf{M}}\left(\boldsymbol{w}_1^\top h(\boldsymbol{x})/\rho\right) < \mathbb{E}_{\mathsf{M}}\left(\boldsymbol{w}_0^\top h(\boldsymbol{x})/\rho\right)$ with high probability and then using standard concentration bound presented in lemma 14, we achieve the desired result.

Since $\|\boldsymbol{x}-\boldsymbol{x}_*\| \leq \frac{\eta}{2}$, all the $\lceil\frac{k+1}{2}\rceil$ nearest neighbors of $\boldsymbol{x}$ have same label. Let $\|\boldsymbol{x}-\boldsymbol{x}_*\|_\infty \leq \Delta/s$, where $\Delta = \frac{1}{2}\left(\tau_{\boldsymbol{x}}(\rho/2m) - \tau_{\boldsymbol{x}}(\rho/m)\right)$. Using lemma 11, we get $q(\boldsymbol{x},\boldsymbol{x}_*) \geq 1/2$. Combining this with part (iv) of lemma 13, we get $\mu_1 = \mathbb{E}_{\mathsf{M}}\left(\boldsymbol{w}_1^\top h(\boldsymbol{x})/\rho\right) \leq 1-q(\boldsymbol{x},\boldsymbol{x}^*) \leq 1/2$.

If $\boldsymbol{x}$ is sampled from a permutation invariant distribution, using corollary 12, we get $\mathbb{E}_{\boldsymbol{x}} q(\boldsymbol{x},\boldsymbol{x}_i) = \rho/m$ for each $\boldsymbol{x}' \in S^0$, and thus using linearity of expectation, $\mathbb{E}_{\boldsymbol{x}}\left(\sum_{\boldsymbol{x}'\in S^0} q(\boldsymbol{x},\boldsymbol{x}')\right) = \sum_{\boldsymbol{x}'\in S^0}\mathbb{E}_{\boldsymbol{x}} q(\boldsymbol{x},\boldsymbol{x}') = \rho n_0/m$. For any $\alpha>0$, using Markov's inequality,

$$pr\left(\sum_{\boldsymbol{x}'\in S^0} q(\boldsymbol{x},\boldsymbol{x}') > \alpha\right) \leq \frac{\mathbb{E}_{\boldsymbol{x}}\left(\sum_{\boldsymbol{x}'\in S^0} q(\boldsymbol{x},\boldsymbol{x}')\right)}{\alpha} = \frac{\rho n_0}{m\alpha}. \tag{20}$$

Specifically, choose $\alpha = 1/4$. Then with probability $\geq 1-\frac{4\rho n_0}{m}$, we have $\sum_{\boldsymbol{x}'\in S^0} q(\boldsymbol{x},\boldsymbol{x}') \leq \frac{1}{4}$. Combining this with part (v) of lemma 13, we immediately get, with probability $\geq 1-\frac{4\rho n_0}{m}$, we have $\mu_0 = \mathbb{E}_{\mathsf{M}}\left(\boldsymbol{w}_0^\top h(\boldsymbol{x})/\rho\right) \geq 3/4$.

Next we show that $\frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho} \leq 3/5$ with high probability.

If we set $\epsilon = \frac{1}{10\mu_1}$, then we get $(1+\epsilon)\mu_1 = \mu_1+\epsilon\mu_1 \leq \frac{1}{2}+\epsilon\mu_1 = \frac{1}{2}+\frac{1}{10} = \frac{3}{5}$. For this choice of ϵ, $\mu_1(1+\epsilon) \leq 3/5$. Therefore,

$$\begin{aligned}\Pr\left(\frac{w_1^\top h(\boldsymbol{x})}{\rho} > 3/5\right) &\leq \Pr\left(\frac{w_1^\top h(\boldsymbol{x})}{\rho} > (1+\epsilon)\mu_1\right) \\ &\leq \exp(-\epsilon^2\rho\mu_1/3) \\ &= \exp\left(-\frac{\rho}{300\mu_1}\right) \\ &\leq \exp\left(-\frac{\rho}{150}\right)\end{aligned}$$

where the first inequality follows from our choice of ϵ, the second inequality follows from Lemma 14, the equality follows from our choice of ϵ and the third inequality follows since $\mu_1 \leq \frac{1}{2}$.

Next, we would like to show $\frac{\boldsymbol{w}_0^\top h(\boldsymbol{x})}{\rho} \geq \frac{5}{8} > \frac{3}{5}$ with high probability. If we set set $\epsilon_1 = \frac{1}{6}$ and using the fact that $\mu_0 \geq \frac{3}{4}$, we get $(1-\epsilon_1)\mu_0 = \frac{5}{6}\mu_0 \geq \frac{5}{6}\cdot\frac{3}{4} = \frac{5}{8}$.

Now we have,

$$\begin{aligned}\Pr\left(\frac{w_0^\top h(\boldsymbol{x})}{\rho} < 5/8\right) &\leq \Pr\left(\frac{w_0^\top h(\boldsymbol{x})}{\rho} < (1-\epsilon_1)\mu_1\right) \\ &\leq \exp(-\epsilon^2\rho\mu_0/2) \\ &= \exp\left(-\frac{\rho\mu_0}{72}\right) \\ &\leq \exp\left(-\frac{\rho}{96}\right)\end{aligned}$$

where the first inequality follows from our choice of ϵ_1, the second inequality follows from Lemma 14, the equality follows again from our choice of ϵ_1 and the third inequality follows from the fact that $\mu_1 \leq 1$.

Therefore, with probability at least $1-\left(\frac{4\rho n_0}{m}+e^{-\frac{\rho}{150}}+e^{-\frac{\rho}{96}}\right)$ we have, (i) $\frac{\boldsymbol{w}_0^\top h(\boldsymbol{x})}{\rho} \geq \frac{5}{8}$, and (ii) $\frac{3}{5} \geq \frac{\boldsymbol{w}_1^\top h(\boldsymbol{x})}{\rho}$. Since $\frac{5}{8} > \frac{3}{5}$, the result follows. □

E. Details on SBFC baseline

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). SimHash is binary like the FlyHash we consider, however, it is not explicitly sparse as FlyHash. In fact, the number of non-zeros in FlyHash is ρ, while for SimHash with dimensionality m, in expectation, we would expect $\approx m/2$ non-zeros in the SimHash. We tune over the SimHash projected dimension m, considering $m<d$ (traditional regime where SimHash is usually employed) and $m>d$ (as in FlyHash). For the same m, SimHash is more costly ($\sim O(md)$ per point) than FlyHash ($\sim O(ms+m\log\rho)$) since $s \ll d$, involving a dense matrix-vector product instead of a sparse matrix-vector one. The dimensionality of

Table 2: Details of a subset of the data sets. For CIFAR-10 and CIFAR-100, we collapse the 3 color channels and then flatten the 32×32 images to points in $\mathbb{R}^{1024}$. For MNIST and Fashion-MNIST, we flatten the 28×28 images to points in $\mathbb{R}^{784}$.

Dataset	n	d	L	Obtained from
Digits	1797	64	10	OpenML
Letters	20000	16	26	OpenML
MNIST	60000	784	10	Tensorflow
Fashion-MNIST	60000	784	10	Tensorflow
CIFAR-10	50000	1024	10	Tensorflow
CIFAR-100	50000	1024	100	Tensorflow

the SimHash m is the hyper-parameter we search over – we consider both projecting down in the range $m \in [1, d]$ (the traditional use) and projecting up $m \in [d, 2048d]$, where d is the data dimensionality.

F. Details on TrainFlyNNFL scaling

We consider the 6 datasets for evaluating the scaling of TrainFlyNNFL (Algorithm 2) with the number of parties τ, when the data is evenly split between all parties. The details regarding the datasets are provided in Table 2.

Raw runtimes. We also present the raw runtimes that were used to generate the speedup plot in Figure 2 in Table 3.

Table 3: Raw runtimes T (in seconds) and speedups S.

Dataset	$T(\tau = 1)$	$T(\tau = 2)$	$S(\tau = 2)$	$T(\tau = 4)$	$S(\tau = 4)$	$T(\tau = 8)$	$S(\tau = 8)$	$T(\tau = 16)$	$S(\tau = 16)$
Digits	3.63±0.06	3.26±0.10	1.11±0.04	1.84±0.06	1.97±0.07	1.36±0.05	2.67±0.11	1.16±0.03	3.12±0.06
Letter	25.72±0.41	14.74±0.42	1.75±0.04	7.72±0.20	3.33±0.11	4.06±0.33	6.38±0.50	2.91±0.07	8.85±0.25
MNIST	1023.59±14.88	518.85±8.19	1.97±0.04	262.68±5.98	3.90±0.08	163.75±5.99	6.26±0.29	122.30±6.80	8.39±0.44
Fashion-MNIST	1410.29±13.66	712.98±3.59	1.98±0.02	360.74±5.47	3.91±0.07	241.70±8.33	5.84±0.18	191.15±1.42	7.38±0.09
CIFAR-10	1300.90±36.63	644.99±4.70	2.02±0.06	330.35±7.21	3.94±0.10	207.57±8.27	6.28±0.33	151.67±3.70	8.58±0.23
CIFAR-100	1268.86±7.85	649.20±5.43	1.95±0.02	333.33±9.36	3.81±0.11	211.20±11.73	6.03±0.33	162.80±1.26	7.79±0.08