SAFE: Machine Unlearning With Shard Graphs

Yonatan Dukler\textsuperscript{1,*}, Benjamin Bowman\textsuperscript{1,2†}, Alessandro Achille\textsuperscript{1,*}, Aditya Golatkar\textsuperscript{1,2†}, Ashwin Swaminathan\textsuperscript{1}, Stefano Soatto\textsuperscript{1}
AWS AI Labs\textsuperscript{1}, UCLA\textsuperscript{2}
\{dukler,bowmaben,aachille,agolatka,swashwin,soattos\}@amazon.com

Abstract

We present Synergy Aware Forgetting Ensemble (SAFE), a method to adapt large models on a diverse collection of data while minimizing the expected cost to remove the influence of training samples from the trained model. This process, also known as selective forgetting or unlearning, is often conducted by partitioning a dataset into shards, training fully independent models on each, then ensembling the resulting models. Increasing the number of shards reduces the expected cost to forget but at the same time it increases inference cost and reduces the final accuracy of the model since synergistic information between samples is lost during the independent model training. Rather than treating each shard as independent, SAFE introduces the notion of a shard graph, which allows incorporating limited information from other shards during training, trading off a modest increase in expected forgetting cost with a significant increase in accuracy, all while still attaining complete removal of residual influence after forgetting. SAFE uses a lightweight system of adapters which can be trained while reusing most of the computations. This allows SAFE to be trained on shards an order-of-magnitude smaller than current state-of-the-art methods (thus reducing the forgetting costs) while also maintaining high accuracy, as we demonstrate empirically on fine-grained computer vision datasets.

1. Introduction

Large-scale neural networks are typically trained on large monolithic datasets. However, real world data often comes from many different sources which may require different treatment depending on their terms of use. In some cases, the terms can change, triggering the need to not just erase a portion of the data, but also remove its influence on the trained model. If the model is trained with the entire dataset in an undifferentiated fashion, even a request to remove a small fraction of the data may result in re-training the entire model on the complement. Considering the scale of large neural networks currently in use, re-training the model after each data erasure is costly. Thus, it is beneficial to develop methods to trace the influence of various segments of data onto the trained model, and to remove their effect if needed, especially as the scale of production models and the amount of training data continues to grow.

One simple and robust approach to forgetting is compartmentalizing the information of different subsets of data into distinct model parameters. In this scenario, the training data is split into disjoint shards, and different parameters or “adapters” are trained separately on each shard, and then ensembled to obtain a final model. The advantage of this approach is that, if the influence of a sample needs to be removed, only the parameters corresponding to its shard have to be retrained. Moreover, if an entire source of data needs to be dropped, one can simply drop all the adapters corresponding to the shards containing its samples. There are, however, two main disadvantages to this approach. On the implementation side, there is an increase in storage/inference scaling with the multitude of adapters to ensemble. On a more fundamental level, since each adapter is trained independently on a fraction of the available data, it forfeits synergistic information that may be present in data stored in different shards. As the number of shards grows, adapters are trained on increasingly impoverished samples, leading to degraded performance relative to monolithic training (see Fig. 2).

Thus, practitioners have to choose a trade-off between increasing the accuracy of the model and minimizing the expected cost of forgetting a sample. In this work, we show that naive sharding, that is uniform partitioning of the training set, is suboptimal in many realistic use cases. Instead, shards should be constructed to maximize synergistic information available at training time. To that end, we introduce SAFE (Synergy Aware Forgetting Ensemble), an algorithm that leverages synergistic information between data shards to maximize accuracy for a given forgetting cost. We show that SAFE allows training on highly sharded data (with as
many as 256 shards) with a 14% accuracy boost over uniform sharding for the same forgetting cost.

To realize this, we introduce the notion of a Shard Graph (SG) (see Fig. 1). Each node of the graph is a small shard of data in the original dataset. (Directed) edges between shards indicate that the corresponding adapter is not trained in isolation as usual, but also uses data from the shards it connects to, thus increasing the accessible synergistic information. On the downside, when a forget request is received, all nodes pointing to the forgotten node must be retrained, increasing the expected forgetting cost. Since different shards have different likelihood of receiving forget requests and contain different information, the problem becomes how to draw connections that maximizes the synergistic gain while minimizing the increase in expected forgetting cost. For example, shards that are unlikely to contain samples that need to be forgotten (e.g., a shard that contains synthetic data or highly vetted data) can be simultaneously connected to many other nodes significantly increasing accuracy without increasing the expected forgetting cost.

A SG may, however, have hundreds of nodes, thus raising the problem of how to train and perform inference with hundreds of adapters. SAFE addresses this problem using InCA adapters [15]. These are small modules connected to a frozen pre-trained backbone, which can be trained in parallel at less than the cost of fine-tuning a single monolithic model. SAFE can thus handle orders of magnitude more shards than other state-of-the-art algorithms. While the use of lightweight adapters may reduce accuracy, we found the loss in accuracy negligible when the chosen backbone is well aligned with the user data, and it is offset by the large reduction in forgetting cost.

We then present two extensions of SAFE. First, we consider the case where information about individual samples from connected shards is bounded using \((\epsilon, \delta)\)-Differential Privacy, which can be interpreted as using a weighted graph with edges of weight \(\epsilon\). In situations where \((\epsilon, \delta)\)-DP is an acceptable guarantee, this formulation can further reduce the expected forgetting cost by avoiding the need to always retrain all connected shards after a forget request. Second, we show that SAFE can be used to serve à-la-carte models [6], where each user can access a model trained only on a user-specific subset data, as long as the connectivity of the SG is compatible with the user access rights.

Empirically, we show that SAFE can reduce the cost of forgetting samples by an order of magnitude compared to similar forgetting algorithms, while maintaining a similar accuracy. While evaluation of forgetting algorithms is usually performed on simple datasets such as CIFAR, we show that SAFE can be applied to more complex and nuanced fine-grained computer vision benchmarks.

2. Related Work

The forgetting/machine-unlearning problem [8, 20] focuses on removing information from a trained machine learning model. One line of work [5, 51, 53, 31, 34, 6] involves splitting the dataset into multiple shards and training separate models for each shard. In such settings forgetting a sample will only affect the model trained on the subset containing the data. In [4] the authors enable forgetting for logistic regression by allowing the model to forget a class. The recent work [51] achieves a 1-class linear classifier trained on only a single class by making predictions based on class-encodings. The authors of [53] propose the model LegoNet which generalizes [5] by modifying the ensembling procedure to a select-and-average approach with the top-\(k\) most relevant sub-models in an instance-dependent manner. Such methods provide instant forgetting by removal of the corresponding model with low re-training time, however, they suffer from increased inference costs directly correlating with the number of shards.

Another line of work involves forgetting for a single deep network. Forgetting is difficult for deep networks [22, 23] due to the highly nonlinear nature of the parameterization and nonconvexity of the learning problem. Such works often use approximations to the non-linear landscape of a deep network to propose unlearning methods. In [28] the authors perform a newton update for unlearning, while in [21, 2] the authors train a linearization of a ResNet-50 starting from a pre-trained model. The linearity of the parameterization, enables convexity of the optimization so that one can forget specific samples and obtain upper bounds on the mutual information after a certain number of forgetting steps. However, such unlearning methods are approximate, and as a result may not completely re-train the model after a fixed number of iterations when the privacy budget is consumed. The works of [50, 45] provide algorithms and seek understanding of unlearning by caching models, and unrolling the iterative steps of the optimization problem.

The unlearning problem has also been explored in federated setups [7, 35, 48, 36, 25] where a client withdraws its data and its influence needs to be removed from the global model and other clients. This approach is different from sharding based methods where there is no notion of a centralized model, and a set of weak models need to be ensembled. Some recent works [38, 29, 11, 46, 43] provide stochastic unlearning algorithms (similar to [1]) with rigorous theoretical guarantees similar to the probabilistic guarantees in differential privacy [16] or algorithmic stability. [29] showed that the “perfect” unlearning algorithms like [5], provide perfect forgetting only for uniform requests, and not adaptive requests. Unlearning guarantees of approximate algorithms decay with the number of forget requests, which require full re-training of the model eventually. In one exten-
A. Forgetting on the Shard Graph

B. Example Shard Graphs

C. InCA Architecture

Figure 1. (A) Forgetting on the Shard Graph. Upon a forget request, the node containing the sample to be forgotten and all sub-models with outbound connections to the node are re-trained. (B) Example Shard Graphs. Prototypical examples of the Shard Graph include (a) Isolated Shards: (instant forgetting of whole shards, low accuracy). (b) Disjoint Sharding: (fast forgetting, competitive accuracy). (c) Safe Shard: one shard not likely to receive forget requests connected to all nodes (instant forgetting, competitive accuracy in appropriate settings). (d) No Sharding: (forgetting is infeasible, ideal performance). (C) InCA Architecture. We use lightweight cross-attention adapters acting upon the representations of a pretrained transformer encoder. Each node in the graph is associated to an InCA adapter trained on the union of the data of a node and all its outbound edges.

3. Forgetting on Shard Graphs

In this section we introduce the notion of a Shard Graph and the flexible forgetting approach it enables. We then show that existing shard-based forgetting algorithms can be reinterpreted as a degenerate case where the graph does not contain any edges. In the next section we will introduce SAFE, a forgetting algorithm that can take full advantage of the provided graph.

Shard Graphs. For ease of notation, in this section we will assume our label space is the discrete set $Y = \{1, \ldots, K\}$. The Shard Graph (SG) is a directed graph $G = (V, E)$ where the set of nodes $V := \{S_1, \ldots, S_n\}$ denotes different data sources or data shards $S_i \subset \mathcal{X} \times Y$, which are defined by the user based on their application. We use a directed edge $(S_i, S_j) \in E$ between two shards (i.e., “$S_i$ points to $S_j$”) to denote that when training the adapter corresponding to $S_i$ we also allow access to data from $S_j$ (see later). In all graphs we consider we assume implicitly that each node $S_i$ has a self connection, $(S_i, S_i) \in E$, that is, the adapter of a shard is always trained on that shard’s data.

Definition of forgetting. Consider an algorithm $\mathcal{A}$ (possibly stochastic like SGD) that, given a shard graph $G$ as input, outputs a model trained on the shards of $G$. In this case, we denote with $\Theta \mapsto \mathbb{P}(\mathcal{A}(G) \in \Theta)$ the probability distribution of possible models produced by $\mathcal{A}$ given a graph $G$. Let $G'$ be a shard graph where some of the data has been removed (for example, removing an entire node/shard or removing samples of the shard). For a training algorithm $\mathcal{A}$, we define a forgetting procedure $U(\mathcal{A}(G), G')$ which takes a trained model $M = \mathcal{A}(G)$ and the reduced graph $G'$ to output a new model $M'$ which is indistinguishable from a model trained directly on $G'$. To incorporate the stochasticity of $\mathcal{A}$, we require that the distribution of models outputted by the forgetting procedure $U(\mathcal{A}(G), G')$ matches $\mathcal{A}(G')$:

$\mathbb{P}(U(\mathcal{A}(G), G') \in \Theta) = \mathbb{P}(\mathcal{A}(G') \in \Theta)$ for all events $\Theta$.

Note that a forgetting algorithm can always trivially satisfy this by ignoring $\mathcal{A}(G)$ and retraining from scratch on the reduced graph $G'$, that is, setting $U(\mathcal{A}(G), G') = \mathcal{A}(G')$. Doing so, however, is expensive especially if $G'$ contains lots of data. The quality of a forgetting procedure is gauged by its ability to minimize the cost of forgetting while maintaining high accuracy.

Independent shard forgetting. Given a shard graph $G$, a trivial but efficient forgetting approach [5, 6, 51] is to discard the edges and train a separate adapter on each node $S_i \in V$. More precisely, let $A(S_i)$ denote the adapter trained on the data in $S_i$. The model corresponding to the graph $G$ is then the ensemble of all the adapters:

$\mathcal{A}(G) = \text{ensemble}(\{A(S_i)\}_{S_i \in V})$. \hspace{1cm} (1)

The ensembling procedure is application specific, for instance, in classification we average the logits of the models.

Such $\mathcal{A}$ admit a simple forgetting procedure $U$. If requested to forget an entire node/shard in $G$, we simply need to drop
the corresponding adapter incurring a constant expected forgetting cost $O(1)$. When requested to forget a sample (or subset of samples) from a shard, we only need to retrain the corresponding adapter on the remaining data. Assuming that the cost of training an adapter is linear in the size of its training data, the expected cost to forget one sample is $O(|S|)$, where $|S|$ is the expected shard size.

When the data is divided into small shards, the expected forgetting cost is a fraction of the cost of retraining the whole model from scratch (all shards). However, as we see in Fig. 2, the accuracy of an ensemble model trained on many small shards (SISA, orange curve) can be significantly lower than the accuracy of a single model trained on all the data simultaneously (right-most point in the curve). This can be attributed to the loss of synergistic information when training independently on many small shards.

4. SAFE

SAFE aims to reduce the impact of the loss of synergistic information by allowing shards connectivity in a way that does not significantly increase the expected forgetting cost while improving performance. Our proposed method can be formulated easily in our proposed shard graph formalism.

Let $G$ be a shard graph. Rather than training an adapter independently on the data of each shard, SAFE trains an adapter using also all the data contained in the connected shards. Formally, the model produced by SAFE is:

$$A(G) = \text{ensemble} \left\{ A(\bigcup N(S_i)) \right\}_{S_i \in V} \quad (2)$$

where $\bigcup N(S_i)$ denotes the union of all the data contained in the shards connected to $S_i$ – that is, its outbound neighborhood $N(S_i)$ (see Fig. 1C). Note that, when the graph $G$ doesn’t have any edges, Eq. (2) reduces to Eq. (1).

Training on the union of data from connected shards exploits the synergistic information among those shards to improve accuracy at the expense of increasing the forgetting cost. However, depending on the structure of the data and the likelihood of forget requests, the increase in accuracy can greatly outweigh the increase in forgetting cost.

**Forgetting with SAFE.** Unlike before, if a sample in a shard is deleted, we need to retrain not only the adapter corresponding to that shard, but also the adapters of all shards pointing to it. The cost of the procedure scales with the total amount of training data that needs to be revisited to train the adapters. Letting $x \in S_i$ be a sample to be forgotten, we can write as

$$M_x := \bigcup \{ U N(S_j): S_i \in N(S_j) \} \quad (3)$$

the total data needed to retrain the adapters of all shards $S_j$ that point to $S_i$. Hence, the expected cost of forgetting depends on the expected size of $M_x$, which in turns depends on the graph topology. We now analyze some interesting cases, which will inform our experimental setup.

**Random connectivity.** Suppose each node is connected to $d$ other nodes uniformly at random. Then in expectation (see Appendix) we have:

$$\mathbb{E}|M_x| = \Theta(|S|d^2).$$

Which scales quadratically with the degree.

**Partition in disjoint cliques.** Consider now the case where the graph is partitioned into disjoint cliques of size $d$ (see Fig. 1B (b)). In this case, all the $N(S_j)$ in Eq. (3) perfectly coincide and the union corresponds to $N(S_j)$. This leads to an expected forget cost that scales with the size $d$ of the shard’s clique

$$\mathbb{E}|M_x| = d \cdot |S|.$$  

In particular, compared to the random connection case, the cost is linear instead of quadratic in the degree of the nodes.

Based on this analysis, we focus on graphs that can be represented as a union of disjoint cliques, as it leads to lower cost of forgetting while allowing the same amount of connectivity. Finally, the case where an entire shard is deleted, rather than a single example, has similar analysis. The corresponding adapter needs to be removed from the ensemble, and all adapters of shards pointing to it need to be retrained. The expected cost scales in the same way as before as a function linear with $d$.

**Refined Shard Graph.** In practice, it may often happen that different shards contain samples from different classes, or even from a single class. To deal uniformly with all
the cases — and to enable finer level of sharding and thus faster forgetting — it is convenient to restrict the adapters of SAFE to binary classifiers for each label present in the adapter’s training set. This can be seen as refining the graph so that each node contains data from only one class (its positive samples) with the negative examples coming from the connected nodes. Formally, given a shard $S \subset \mathcal{X} \times \mathcal{Y}$ we define the label occurrence map $L(S) = \{ y : (x, y) \in S \}$. Furthermore, for each label $k \in \mathcal{Y}$, we define the refined shard $S^{(k)} = \{ (x, y) \in S : y = k \}$. Then given a shard graph $G = (V, E)$ we construct a refined vertex set

$$V' := \{ S^{(k)} : S \in V, \quad k \in L(S) \}$$

and edge set

$$E' := \{ (S^{(k)}, S^{(k')}) : (S, S') \in E, h \in L(S), k \in L(S') \}.$$ 

Hence, for each node $S^{(i)} \in V'$ we train a binary classifier where the positive examples come from $S^{(i)}$ and the negative examples come from the all the connected nodes.

5. Synergistic Bilevel Sharding

We now tackle the question of how to generate a graph structure that increases the synergistic information for a given expected forgetting budget. Forgetting methods usually split the data in uniform shards. However, we note that samples from the same classes, or related classes, have more synergistic information, so should preferentially appear in the same clique, even if this means that the clique may not contain all classes due to size constraints (note that, as mentioned before, we do not need the adapters to see all classes since each is a binary one-vs-all classifier). Following this intuition, we suggest the following strategy: We first split the data in $n_c$ disjoint “coarse” shards using class-balanced subsampling. For each coarse shard, we draw the nodes corresponding to each class and partition them randomly into cliques of size $d$, i.e., each clique contains examples from $d$ classes. This results in a number $n_f = \frac{n_{classes}}{d}$ of “fine” cliques per coarse shard, for a total of $n = n_c \cdot n_f$ cliques. We note that the expected forgetting costs scales with $1/n$, i.e., the amount of data in each clique. Partitioning the data uniformly, as commonly done, is equivalent to selecting $n_f = 1$.

**Optimal graph structure.** Since the expected forgetting cost is the same as long as the product $n = n_c \cdot n_f$ remains constant, we check what $n_f$ produces the best model for a fixed $n$. Low $n_f$ increases the variety of the classes in each shard at the expense of loss of synergistic information, while high $n_f$ increases the synergistic information but each model sees a smaller subset of classes. In Fig. 3 we plot the trade-off on the MIT-67 dataset. We see that indeed uniform sharding ($n_f = 1$) is never optimal, and that using a higher (but not too high) values of $n_f$ is essential to get good accuracy when aiming for a low forgetting time, and can improve accuracy by more than 15%, thus supporting the design choices for our method. Random partitioning across coarse and fine levels also makes our method robust against non-adaptive forget requests.*

6. Efficient implementation of SAFE

Training independent large models on hundreds of shards leads to prohibitive storage and inference costs, and increases the expected cost of retraining for each forget request. This suggest using a shared backbone and a series of small shard-specific adapters. We use a variant of Open-InCA cross-attention adapters [15], which lend themselves easily to massive parallel training and inference (See Appendix A for details) while also providing high-accuracy on difficult fine-grained classification tasks. With Open-InCA adapters, we can efficiently compartmentalize the training data of each shard to its corresponding Open-InCA class parameters.

**InCA adapters [15].** Let $z = (z_1, \ldots, z_n) = f_w(x)$ be the activation map produced by a frozen vision encoder $f_w(x)$. Given a set of learnable class-specific query tokens $q = (q_1, \ldots, q_K)$, the output $y = CA \theta(z, q, v)$ of InCA adapters $CA$ are the logits $y = (y_1, \ldots, y_n)$ defined by

$$y_i = v_i \cdot e_i$$

$$e = \text{cross-attention}_\theta(z, q)$$

where $\theta$ are the parameters of the cross-attention layer and $v = (v_1, \ldots, v_k)$ is a learnable set of vectors $v_k$ (which can be interpreted as binary linear classifiers). To keep the notation uncluttered, in the following we write $q_i$ to denote

*When the graph structure is data dependent, DP techniques may be required to prevent information leakage from it.
both the query vectors and the corresponding classification vector $v_i$. An important property of Open-InCA for our application is compositionality. Let $[q, q']$ denote the concatenation of $q$ and $q'$, then

$$
CA_\theta(z, [q, q']) = [CA_\theta(z, q), CA_\theta(z, q')].
$$

(4)

This suggests that, rather than training a separate model on each node, we can train individual node specific $q_i$ together by concatenating them to obtain the final model. In addition to the compositionality by learning different queries $q_i$ for each classifier, Open-InCA remains expressive in selecting sub-task specific representations for each classifier.

**Applying InCA to SAFE.** We freeze and use the same cross attention weights $CA_\theta$ which are shared across all nodes, with frozen parameters $\theta$. Then, for each reduced shard $S_k$, we compute corresponding queries $q^k_i$, and train the resulting binary one-versus-all classifier

$$
y^k_i(z) = CA_\theta(z, q^k_i)
$$

on the data of all connected shards using a binary cross entropy (BCE) loss. If there is a clique of nodes in the refined shard graph, we can group them together and train simultaneously instead with the cross entropy loss. At inference time, we utilize the compositionality of the $CA_\theta$ adapters, by computing the concatenation of all $q^k_i$ sharing the computation of all the adapters in a single forward pass and easily ensemble the resulting logits, leading to the final logits,

$$
y_i(z) = \text{mean}_k \left( CA_\theta(z, q, v) \right).
$$

In the equation above the embedding $z = f_w(x)$ is of a test sample $x$. For classification, we select the class corresponding to the highest logit. Training each $q^k_i$ and $v^k_i$ sequentially is still expensive, and wastes computation since the same sample can be used as a negative to train multiple connected nodes. Instead, since for the BCE loss the gradients of different $q_i$ are independent, we can train all the $q_i$ at the same time over a single epoch on the whole (re)-training set, provided the loss is appropriately masked to prevent information from unconnected shards from transpiring into each $q_i$ (see Appendix A for details). Further since InCA does not backpropagate through the backbone $f_w$, we can also pre-compute and store the embeddings $z$. Using all these techniques, we are able to train hundreds of shards at the same time in under 10 minutes on a single GPU.

7. **Prototypical classifier**

Class prototypes are another viable approach to forgetting used by ARCANET [51]. Given a dataset $\{ (x_i, y_i) \}_{i=1}^N$ and an embedding $z = f_w(x)$, we define the prototype of the $k$-th class as

$$
p_k = \frac{1}{N_c} \sum_{(x,y) \in D^{(k)}} f_w(x)
$$

where $D^{(k)}$ are the samples of class $k$ and $N_c = |D^{(k)}|$. We can then construct a simple linear classifier

$$
y_k(z) := d_{cos}(z, p_k),
$$

where $d_{cos}$ denotes the cosine distance. Such “prototypical” classifiers allow instantaneous forgetting: to forget a training sample $(x_i, y_i)$ we just need to remove it from its class prototype $p_{y_i} \rightarrow \frac{N_c p_{y_i} - f_w(x_i)}{N_c - 1}$. On the other hand, this classifier has suboptimal classification accuracy compared to a trained classifier on large shards of data.

However, when the shards consists only of a few samples, classifiers trained on individual shards may overfit. In such cases, the prototypical classifier can be used to provide an inductive bias [44]. Since the added computational and space complexity to use the prototypical classifier is negligible, we combine it into the SAFE model using the following expression:

$$
\text{SAFE}(z) = (1 - \lambda) \cdot M(G)(z) + \lambda \cdot \text{Proto}(z),
$$

where $M(G)$ is the ensemble model in Eq. (2), Proto denotes the prototype-based classifier and $\lambda = \exp \left( -\frac{d|S|}{100} \right)$ is an interpolation weight that relies more on the prototypical classifier when the amount of data $d|S|$ used to train each adapter is small.

8. **Extensions of SAFE**

Stochastic forgetting for reduced cost. In the previous sections we presented forgetting approaches that use the edges in the shard graph to define complete usage of a shard or no usage (when there is no connection). Below we present the notion of limited shard information defined via differential privacy (DP), in which data of different nodes (data shards) of the graph are shared to a sub-model in a differentially private fashion. This can be defined by an edge weight that bounds the information shared about each sample, and can be interpreted as the probability a sample is identified in the training set. Specifically consider the binary classifier sub-models introduced in Section 6, for the positive-negative samples defined by the graph topology we assign complete usage to the positive node and limited usage to the bound connections of the negative nodes. This corresponds to a special case of mixed DP [21], which results in a simple training algorithm (SAFE-DP) where during each epoch, each node is trained with its own data without privacy, and with data from the neighbouring node using DP. This algorithm satisfies an approximate definition

---

<sup>1</sup>We find that simply using a random initialization for $\theta$ provides a good and unbiased performance across tasks.
of forgetting, more precisely, $A(G)$ is called an $(\alpha, \beta)$-unlearning algorithm if
\[
P(U(A(G), G') \in E) \leq e^{\alpha}P(A(G') \in E) + \beta
\]
for all events $E$. This definition measures the privacy leakage in terms $(\alpha, \beta)$ using group DP. We enable each user to specify their target $\alpha_{\text{target}}$ (such that $\beta < 1$), and employ privacy accounting to identify budget overflow ($\beta > 1$) for sequential forget requests. Such an algorithm is useful in adversarial conditions for protection against worst case adaptive forget requests.

**À-la-carte models via the Shard Graph.** The problem of constructing a unique model for a particular user that only uses data consistent with their access permissions and personal preferences, i.e., the “à-la-carte learning” problem [6], can be solved using SAFE. For a given user one can identify which nodes in the graph the user is unable or unwilling to access. Based on this, one can ensemble only the adapters trained on the nodes with no outbound connections to the ineligible nodes. Alternatively, one could simulate an artificial “forget request” by dropping all the data that the user is unable to access and retraining the corresponding InCA adapters. Since InCA adapters with cached activations can be trained in seconds, one could potentially perform this training efficiently to serve custom models to different users on-demand.

### 9. Experiments

**Model.** In our experiments we use as encoder a ViT-L/16 transformer architecture [14] pretrained on ImageNet-21k at 224 image resolution.\(^1\) We use the InCA adapters from [15], in particular we train the head and queries, but keep the cross-attention layer frozen and shared between all shards. We train each adapter with AdamW for 30 epochs using cosine annealing starting from $\eta = 0.05$ and weight decay of $10^{-4}$. See the Appendix for full details.

**Datasets.** We evaluate on the following vision classification datasets, covering fine-grained classification tasks and domain shifts with respect to the ImageNet pretraining: CUB-200 [47], MIT-67 [41], Caltech-256 [27], CIFAR-100 [33], Describable Textures (DTD) [13], Stanford Cars [32], and Stanford Dogs [30].

**Baselines.** We compare our method against two main classes of methods for forgetting: Prototypes (motivated by ARCANE\(^3\) [51]) uses a prototype based classifier (Section 7), which allows constant time forgetting but which cannot be trained to improve accuracy, SISA [5] performs forgetting by training an ensemble of models on shards created by sampling uniformly at random. This corresponds to using $n_f = 1$ when structuring the bilevel sharding (Section 5). Like in SAFE, increasing the number $n$ of shards leads to lower forgetting time but also lower accuracy. We re-implement SISA using the same architecture and training scheme as SAFE for a direct comparison. [5] uses further slicing of each shard (through intermittent model checkpoints) to further speed-up re-training complementarily. Slicing can be similarly incorporated in SAFE, but we isolate slicing from our analysis as it is an orthogonal and complementary approach that can be applied in each method (with increased storage costs).

### Table 1. Accuracy of Unlearning Approaches

<table>
<thead>
<tr>
<th>Dataset/Method</th>
<th>No Sharding</th>
<th>Prototypes</th>
<th>8</th>
<th>64</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>SAFE</td>
<td>SISA</td>
<td>ProtoSISA</td>
</tr>
<tr>
<td>Caltech-256</td>
<td>94.3%</td>
<td>93.2%</td>
<td>94.0%</td>
<td>93.6%</td>
<td>93.6%</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>83.1%</td>
<td>71.2%</td>
<td>84.1%</td>
<td>84.0%</td>
<td>84.0%</td>
</tr>
<tr>
<td>CUB-200</td>
<td>88.3%</td>
<td>85.8%</td>
<td>86.1%</td>
<td>83.9%</td>
<td>83.9%</td>
</tr>
<tr>
<td>DTD</td>
<td>77.8%</td>
<td>73.8%</td>
<td>77.1%</td>
<td>76.1%</td>
<td>76.1%</td>
</tr>
<tr>
<td>MIT-67</td>
<td>87.9%</td>
<td>85.8%</td>
<td>86.9%</td>
<td>86.9%</td>
<td>86.9%</td>
</tr>
<tr>
<td>Stanford Cars</td>
<td>75.7%</td>
<td>41.0%</td>
<td>62.1%</td>
<td>55.8%</td>
<td>55.8%</td>
</tr>
<tr>
<td>Stanford Dogs</td>
<td>87.9%</td>
<td>88.0%</td>
<td>89.2%</td>
<td>89.2%</td>
<td>89.2%</td>
</tr>
<tr>
<td>Average Acc.</td>
<td>85.0%</td>
<td>77.0%</td>
<td>82.8%</td>
<td>81.4%</td>
<td>81.4%</td>
</tr>
</tbody>
</table>

\(^1\)We use the `vit_large_patch16_224_in21k` pre-trained model in the timm library [49].

\(^3\)In [51] they work with smaller models (e.g. ResNet-18 with 11M params.) and train a separate embedding for each class, whereas we take the embeddings of a fixed large pretrained transformer (305M params.).
average trade-off between accuracy and forgetting time for SAFE and other methods from the literature (SISA [5] and Prototypes [51]) across the datasets in Table 1 for various target forgetting times. We see that SAFE performs uniformly better than SISA and Prototypes for all forgetting budgets. Thanks to the ability to train, it outperforms Prototypes for higher forgetting time budgets, while it significantly outperform SISA in the low-forgetting-time regime (high-sharding) due to better use of synergistic information.

**Domain shift.** In Table 1 we see that the accuracy of SAFE and other forgetting methods decreases more rapidly with the sharding level for datasets that have a significantly different distribution from the ImageNet-21k pretraining of the backbone (e.g., DTD and Stanford Cars). We hypothesize that this is because the synergistic information of different samples is already contained in the backbone when the data is similar to the pre-training, hence the loss due to sharding is less influential. However, thanks to its better handling of synergistic information, we see that SAFE performs significantly better than the other baselines on difficult domains.

**Ablations.** We can ask whether the better performance of SAFE is due to the use of prototypes, the synergistic information or both. We have seen in Figure 3 that synergistic information alone improves over SISA ($n_f = 1$). In Fig. 2 we also show the result of a further baseline, ProtoSISA, obtained by adding a prototype classifier to SISA using the same weighting scheme used for SAFE. We see that while adding prototypes to SISA boosts performance, SAFE still significantly outperforms both other methods, showing that indeed both aspects are important.

**Instant forgetting.** In certain situations, a service provider will need to satisfy a forget request instantly and thus will need to drop an entire data source without retraining, meaning that all adapters using those samples will need to be dropped without being retrained and replaced. To investigate how robust SAFE, SISA, and ProtoSISA are to such requests, in Fig. 4 we plot the relative accuracy of SAFE, SISA, and ProtoSISA after a series of forget requests for ensembles with 256 shards. We see that SAFE exhibits a smaller decline in relative accuracy in the presence of forget requests, and uniformly outperforms SISA and ProtoSISA. For all methods the marginal decline in accuracy increases as the number of forget requests increases, suggesting that the ensemble becomes more sensitive as additional predictors are dropped.

**Stochastic forgetting.** In Section 8 we propose a DP-based mechanism that allows shards to receive up to a given number $k$ of forget requests without having to retrain a whole clique, at the expense of decreased accuracy due to the bound on information imposed by DP. In Fig. 5 we show this trade-off on a dataset with significant distribution shift (Stanford Cars). The model trained, for example, with SAFE-DP ($k = 8$) provides lower accuracy than SAFE ($n = 8$) (51.0% instead of 62.1%), but has a better worst case guarantee (it can accommodate 8 sequential forget requests before retraining). Thus SAFE-DP may trade-off a worst case privacy guarantee for model accuracy. This is especially useful in settings with adversarial forget requests [29].

10. Multi-Domain synergy experiments

Next we study SAFE’s ability to harness synergistic effects for tasks involving multiple domains. For this we consider the 4-domain challenge, DomainNet-126 [42] that is a curated subset of the extended DomainNet suite [40]. To best evaluate synergy between domains, we ensure each domain’s dataset is of the same size, and we sub-sample the training and test set of each domain to be of fixed size and with a balanced distribution of classes. This results in 6300 training samples and 2520 test samples for each domain.

To test the benefits of SAFE for the multi-domain task of predicting among the 126 different common categories,
now with images coming from different domains, we consider different SG topologies. We define the topologies tested as follows.

- **SISA In-Domain** Each domain is trained separately with SISA-style uniform sharding with 16 shards in each domain (64 total shards).

- **SISA Cross-Domain** The training datasets are merged into a single cross-domain training dataset that is used for SISA-style uniform sharding (shards are maintained to be of same size as in “SISA In-Domain”, e.g. 64 total shards).

- **SAFE In-Domain** SAFE is applied to each domain separately, which creates cliques containing subsets of classes with each clique having the same re-training costs as the previous two approaches.

- **SAFE Cross-Domain** The cliques are constructed to contain synergistic connections between different domains with the overall clique size remaining the same.

In Table 2 we evaluate the different approaches for a 64 shard level. For in-domain topologies this corresponds to 16 shards per-domain. For SAFE we use 4 coarse shards and 4 fine shards. We observe that modifying the Shard Graph topology for SAFE and allowing for shard connections between different domains, SAFE is capable of learning more accurate representations at the same training and unlearning costs.

### Table 2. Multi-domain unlearning on DomainNet-126

<table>
<thead>
<tr>
<th>Method</th>
<th>All</th>
<th>Clipart</th>
<th>Sketch</th>
<th>Real</th>
<th>Painting</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAFE (synergy)</td>
<td>76.11</td>
<td>69.84</td>
<td>62.02</td>
<td>86.67</td>
<td>61.35</td>
</tr>
<tr>
<td>SAFE (synergy)</td>
<td>76.11</td>
<td>69.84</td>
<td>62.02</td>
<td>86.67</td>
<td>61.35</td>
</tr>
<tr>
<td>SAFE (synergy)</td>
<td>76.11</td>
<td>69.84</td>
<td>62.02</td>
<td>86.67</td>
<td>61.35</td>
</tr>
<tr>
<td>SAFE (synergy)</td>
<td>76.11</td>
<td>69.84</td>
<td>62.02</td>
<td>86.67</td>
<td>61.35</td>
</tr>
</tbody>
</table>

In Table 2 we evaluate the different approaches for a 64 shard level. For in-domain topologies this corresponds to 16 shards per-domain. For SAFE we use 4 coarse shards and 4 fine shards. We observe that modifying the Shard Graph topology for SAFE and allowing for shard connections between different domains, SAFE is capable of learning more accurate representations at the same training and unlearning costs.

### 11. Conclusion

We introduced the Shard Graph, a directed graph describing the access relations between data sources for training. This graph informs the creation of SAFE: an ensemble of lightweight adapters trained on the datasets specified by the graph structure. By constructing the graph to maximize synergies between datasets while minimizing connections and retraining time, we are able to handle an order of magnitude more shards than competing methods, and achieve a 14% accuracy boost over competing methods for the same forgetting cost. We conclude that maximizing synergistic information while minimizing dataset overlap is the fundamental trade-off at the core of compartmentalization-based forgetting, which so far has been under-explored. In some cases, the accuracy of our method may be limited by the use of light-weight adapters. However, InCA adapters demonstrate high accuracy while allowing us to efficiently train many adapters in parallel without information leakage and permit fast retraining, which make them conducive to compartmentalization-based forgetting. Rather, we find that the bottleneck in accuracy is mainly due to the loss of synergistic information due to sharding which we alleviate with SAFE. Finally, optimizing the shard graph structure to utilize additional properties in the data without leaking sensitive information is an important problem which we leave to future work.

### References


[31] Korbinian Koch and Marcus Soll. No matter how you slice it: Machine unlearning with sisa comes at the expense of minority classes. In First IEEE Conference on Secure and Trustworthy Machine Learning. 2

[32] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In 4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13), Sydney, Australia, 2013. 7, 19


Supplementary Material

A. Parallel and compartmentalized training of SAFE

In the following section we review the Open-InCA architecture and its use for efficient training of compartmentalized adapters in SAFE. In our training we are able to train all adapters in parallel while having exact control of each parameter’s access to samples’ gradients. This approach enables us to define the Shard Graph flexibly and provide custom “data access” to each adapter while training all of SAFE’s adapters in parallel.

Open-InCA. In the work of [15] it is shown that lightweight cross-attention adapters attached to intermediate network representations are capable of extracting relevant representations and achieving competitive accuracy relative to expensive full fine-tuning. For regular learning, given an activation map \( z \) of a pre-trained model, the InCA adapter structure is defined as

\[
e = \text{cross-attention}_{\theta_1}(z, [q])
\]

\[
y = \text{Linear}_{\theta_2}(e).
\]

By learning the cross-attention layer parameters and the query \( q \), the adapter is capable of extracting task-relevant representations from the activation map \( z \).

For more flexible learning scenarios the authors present Open-InCA, where each class prediction is computed via a separately learned query and class-head parameters,

\[
e = \text{cross-attention}_{\theta_1}(z, [q_1, \ldots, q_C])
\]

\[
y = \text{Diag-Linear}_{\theta_2}(e).
\]

Here \( \text{Diag-Linear}_{\theta_2} \) is a mapping that takes the query embeddings \( [e_1, \ldots, e_C] \) and acts on them “diagonally” through the classification vectors \( [v_1, \ldots, v_C] \):

\[
y_i := v_i \cdot e_i.
\]

We follow the Open-InCA approach which is originally defined for class incremental learning, but for our settings we would like to achieve complete isolation of the learning of each adapter.

Adapter isolation. To achieve isolated learning of each adapter we apply Open-InCA’s “query only learning” [15]. This amounts to not optimizing the cross-attention layer weights \( \theta_1 \) and only training \( [q_1, \ldots, q_C] \) and the separate classification head parameters \( [v_1, \ldots, v_C] \). Nonetheless, applied with the traditional \( \text{softmax} \) and Cross-Entropy loss, query only training still leaks information of each sample in the batch to each adapter. Recall that \( \text{softmax} \) is defined as

\[
[\text{softmax}(y)]_i = \frac{\exp(y_i)}{\sum_{k=1}^{C} \exp(y_k)}.
\]

The \( \text{softmax} \)'s gradient of each class prediction will be affected by the predictions of all other adapters (due to the denominator), thereby leaking information. To rectify this, we replace the \( \text{softmax} \) with a sigmoidal mapping

\[
\sigma(y_i) = \frac{1}{1 + \exp(-y_i)},
\]

so the loss of each adapter prediction relies solely on the adapter’s logit value \( y_i \). Using the sigmoidal mapping enables us to use SAFE in more flexible learning settings. This is because now each node in the shard graph can be responsible for learning just a single binary classifier for a particular class.

Transitioning to Open-InCA with sigmoidal loss results in hundreds or even thousands of individually learned binary classifiers. Trained sequentially the vast number of adapters will result in prohibitively long training time.

Loss masking. Instead we show that we can learn all of the adapters within an Open-InCA adapter in parallel while still isolating information via loss masking. As stated in Eq. 4 of the manuscript the different queries of the Open-InCA

![Figure 6. SAFE Compartmentalized training with Open-InCA](image)

We illustrate the use of Open-InCA for efficient and parallelized training with SAFE. (A) In Open-InCA, each learned binary classifier corresponds to its own dedicated query (yellow) and class-head (orange), in blue is the shared frozen cross-attention module. (B) Given a data-sample (green), we may control which adapters can be updated with its gradient, and based on the architectural choice there will be no information leakage. (C) Using a batch of data-samples and a graph topology of data access rights, we can learn all adapters in parallel using loss masking which automatically blocks and routes gradients to their appropriate adapters.
adapter can be de-composed as
\[
CA_\theta(z, [q, q']) = |CA_\theta(z, [q]), CA_\theta(z, [q'])|.
\]

By using the sigmoid mapping we compute the loss for each Open-InCA binary-class adapter as
\[
L_{i,j} = -y_j^{(i)} \cdot \log(\sigma(CA_\theta(z, [q_i])))
- (1 - y_j^{(i)}) \cdot \log(1 - \sigma(CA_\theta(z, [q_i])),
\]
where \(y_j^{(i)}\) is 1 if example \(j\) is a positive example for the class corresponding to adapter \(i\), and 0 otherwise. For a batch of \(B\) samples and a set of \(M\) adapters, we obtain the loss matrix \(L \in \mathbb{R}^{M \times B}\). Each loss entry \(L_{i,j}\) is computed with frozen cross-attention parameters and the gradients are based on the learnable parameters of only the \(i\)th adapter (due to \(\sigma\)):
\[
\frac{\partial L_{i,j}}{\partial q_k} = 0 \text{ for } k \neq i.
\]

This implies any summation with \(\{a_i\} \subseteq \mathbb{R}^M\)
\[
\ell = \sum_{k=1}^{M} a_k L_{k,j}
\]
has the property of reducing to a single term after differentiation
\[
\frac{\partial L}{\partial q_i} = a_i \cdot \frac{\partial L_{i,j}}{\partial q_i}.
\]

This is highly convenient in automatic differentiation packages, as it reduces our entire optimization (optimizing each adapter) into a single “backwards” call with the summed loss \(\ell\). In particular, for a given SG topology we can define \(A\) for a data batch with \(A \in \{0, 1\}^{M \times B}\) such that
\[
A_{ij} = \begin{cases} 
1 & \text{if shard } i \text{ has access to data sample } j \\
0 & \text{otherwise}
\end{cases}
\]

With this we can optimize the entire SAFE ensemble in parallel in a compartmentalized manner by directly optimizing the objective \(\ell = \sum(L \odot A)\) where \(\odot\) denotes the Hadamard product and \(\sum\) is computed over all the matrix entries.

**DP-SAFE.** For the mixed-privacy DP-SAFE method we can follow a similar process in which we compute a private loss and a non-private loss using 2 separate masking rules defined via \(A_{pp} \in \{0, 1\}^{M \times B}\) and \(A_{direct} \in \{0, 1\}^{M \times B}\) accompanied by respective optimizers DP-SGD and AdamW.

**B. Additional Ablations**

**Additional Sharding Scales.** In Table 5, Table 6, and Table 7 we report the accuracies of SAFE, SISA, and ProtoSISA across a larger range of sharding scales, namely 2, 4, 8, 16, 32, 64, 128, 256. We see that SAFE outperforms SISA on average across all sharding scales, with the gap peaking at 14.3% at 256 shards. Furthermore while the ProtoSISA method improves over SISA at the large sharding scales \(\geq 64\), it still underperforms SAFE by a margin of (3.8% – 10.1%).

**SAFE without prototypes.** SAFE benefits from both synergy-aware sharding and the inductive bias of the prototype model. To separate out these two factors, we consider the SAFE method without prototypes, which we call “NoprotoSAFE”. In Fig. 8 we compare the accuracy of SAFE, NoprotoSAFE, and SISA. We see that at the large sharding scales, the use of prototypes in SAFE gives a modest boost in test accuracy 0.5 – 2.5% over NoprotoSAFE. We see that NoprotoSAFE still significantly outperforms SISA, with margins as high as 10 – 30%, particularly at the large sharding scales \(\geq 64\). We conclude that the synergy-aware sharding of the SAFE method is the primary factor for the boost in performance over SISA.

**SAFE and SISA using a linear model.** Another efficient approach to adapting large pretrained models is to perform head-only finetuning, where only a linear classifier head is trained. Thus a natural question is how do SAFE, SISA, and ProtoSISA perform when replacing the InCA adapters with linear models. In Fig. 7 we report the gain in test accuracy when using the InCA model relative to a linear model at different sharding scales for the various methods. We see that for the SAFE method the InCA model uniformly outperforms the linear model at all sharding scales. For SISA and ProtoSISA, at the larger sharding scales \(\geq 64\) the linear model starts to outperform in some cases. We suspect that SISA and ProtoSISA have difficulty training the cross-attention module when there are very few examples per shard, whereas the synergistic sharding of SAFE enables it to successfully train a more complex model.

**C. Analysis of the Shard Graph**

In Section 4, we theoretically analyze the expected forgetting costs of SAFE for different shard graph topologies. To understand the effect of different topologies, we consider a graph with a determined connectivity level, defined by assigning each node in \(V\) an outbound edges. When presenting SAFE, we concluded that when using a disjoint clique structure for the Shard Graph the expected cost to forget a sample \(x\) (i.e. the number of samples required for retraining) is \(E|M_x| = d \cdot |S|\). We note this is the optimal case and if the degree \(d\) connectivity is applied uniformly at random, \(E|M_x|\) is in fact an order of magnitude larger \(E|M_x| \sim d^2 \cdot |S|\). Below we present a formal statement and proof that \(E|M_x| \sim d^2 \cdot |S|\) for a graph with random connectivity.

**Theorem 1.** Suppose we have a set of nodes \(V = \)
Figure 7. **InCA vs. Linear.** We report the difference in test accuracy between using the InCA model and using a linear model for the methods SAFE, SISA, and ProtoSISA at sharding scales 1, 2, 4, 8, 16, 32, 64, 128, 256. Positive values correspond to InCA having higher accuracy, negative values correspond to the linear model having higher accuracy. Each box corresponds to 7 values, specifically the test accuracy over the 7 datasets we consider. The boxes corresponding to SAFE, SISA, and ProtoSISA are colored green, red, and blue respectively.
\{S_1, \ldots, S_n\} where each \(S_i\) is a source of data and \(S_i \cap S_j = \emptyset\) for \(i \neq j\). Furthermore assume that all the sources are of the same size, i.e. \(|S_1| = |S_2| = \cdots = |S_n|\). Suppose each node is assigned \(d\) outbound edges independently and uniformly (in addition to the default self-connection). Furthermore assume that \(d^2 \leq C|V|^{1/2}\) for some \(C > 0\). Then the expected number of samples \(|M_x|\) needed for retraining upon a forget request for a sample \(x\) is

\[
E[|M_x|] = \Theta(|S_1|d^2).
\]

**Proof.** Assume we receive a forget request for a sample \(x \in S_1\). By uniformity without loss of generality we may assume that \(i = 1\). We will let \(Z\) be a random variable denoting the number of samples we must retrain on whenever receiving a forget request for a sample \(x \in S_1\) where the randomness is taken over the selection of the edges \(E\) of the graph. For a node \(n\) let \(N_{in}(n) = \{v : (v, n) \in E\}\) denote the inbound neighborhood of \(n\) and let \(N_{out}(n) = \{v : (n, v) \in E\}\) denote the outbound neighborhood of \(n\). Then we have that

\[
Z = \left| \bigcup_{n \in N_{in}(S_1)} \bigcup_{v \in N_{out}(n)} v \setminus \{x\} \right|.
\]

We note then that

\[
Z \leq \sum_{n \in N_{in}(S_1)} \sum_{v \in N_{out}(n)} |S_1| \leq |N_{in}(S_1)|(d+1)|S_1|.
\]

Therefore

\[
E[Z] \leq |S_1|(d+1)E[|N_{in}(S_1)|]
\]

We note that for each node \(v \in V \setminus S_1\) that \(I[v \in N_{in}(S_1)]\) is a Bernoulli random variable taking the value 1 with probability \(\frac{d}{|V|-1}\). Furthermore by default we know that \(S_1 \in N_{in}(S_1)\). Therefore

\[
|N_{in}(S_1)| - 1 = \sum_{v \in V \setminus S_1} I[v \in N_{in}(S_1)]
\]

is a sum of \(|V| - 1\) independent Bernoulli random variables. Thus \(|N_{in}(S_1)| - 1\) obeys the Binomial distribution \(Bin(|V| - 1, \frac{d}{|V|-1})\) which has mean \(d\). Thus

\[
E[|N_{in}(S_1)|] = d + 1
\]

and we conclude that

\[
E[Z] \leq |S_1|(d+1)^2 = O(|S_1|d^2).
\]

Now we will prove the lower bound. We note for \(d = 1\) the statement is trivial so we might as well assume \(d \geq 2\). Since we are seeking a lower bound we may assume that after the forget request is received the entire source \(S_1\) is dropped, as this only decreases the number of samples needed to retrain. In this case

\[
Z = \left| \bigcup_{n \in N_{in}(S_1) \setminus S_1} \bigcup_{v \in N_{out}(n) \setminus S_1} v \right|.
\]
We will focus on estimating the conditional expectation

$$\mathbb{E}[Z \mid |N_{in}(S_1) \setminus S_1| = k].$$

We note that by uniformity that the distribution of $Z$ depends only on the size of $N_{in}(S_1) \setminus S_1$ and not the specific collection of nodes in $N_{in}(S_1) \setminus S_1$. Thus we will fix a choice \(\{n_1, \ldots, n_k\}\) of $k$ nodes for $N_{in}(S_1) \setminus S_1$. Let $E$ denote the event that $N_{in}(S_1) \setminus S_1 = \{n_1, \ldots, n_k\}$ and let $P_E(\cdot) := P(\cdot | E)$ denote the probability of an event conditioned on $E$. We note then that

$$\mathbb{E}[Z \mid |N_{in}(S_1) \setminus S_1| = k] = \mathbb{E}[Z | E].$$

Let $A_i = N_{out}(n_i) \setminus S_1$ for $i = 1, \ldots, k$. We note that if $A_1, A_2, \ldots, A_k$ are disjoint then $Z = kd \cdot |S_1|$. Thus we have that

$$\mathbb{E}[Z \mid E] \geq kd \cdot |S_1| \cdot P_E(A_i \cap A_j = \emptyset \ \forall 1 \leq i \neq j \leq k).$$

Thus we proceed to lower bound

$$P_E(A_i \cap A_j = \emptyset \ \forall 1 \leq i \neq j \leq k).$$

The number of disjoint choices of $A_1, A_2, \ldots, A_k$ is

$$\prod_{j=0}^{k-1} \left( |V| - (k + 1) - j(d - 1) \right).$$

The number of total choices of $A_1, A_2, \ldots, A_k$ is

$$\binom{|V| - 2}{d - 1}.$$ 

Thus we have

$$P_E(A_i \cap A_j = \emptyset \ \forall 1 \leq i \neq j \leq k) \geq \left[ \frac{|V| - (k + 1) - (k - 1)(d - 1)}{d - 1} \right]^k.$$ 

Now assume that $k \in [(1 - t)d, (1 + t)d]$ for some fixed $t \in (0, 1)$. Note then that

$$\left[ \frac{|V| - (k + 1) - (k - 1)(d - 1)}{d - 1} \right]^k \geq \left[ \frac{|V| - kd}{|V|} \right]^k.$$ 

Thus it follows for $k \in [(1 - t)d, (1 + t)d]$ we have that

$$\mathbb{E}[Z \mid |N_{in}(S_1) \setminus S_1| = k] \geq |S_1| d \cdot k \cdot P_E(A_i \cap A_j = \emptyset \ \forall 1 \leq i \neq j \leq k) = \Omega((1 - t)|S_1|d^2).$$

We note it then suffices to show that for some fixed $t \in (0, 1)$

$$P(|N_{in}(S_1) \setminus S_1| \in [(1 - t)d, (1 + t)d]) = \Omega(1).$$

We recall that for each node $v \in V \setminus S_1$ that $\mathbb{I}[v \in N_{in}(S_1)]$ is a Bernoulli random variable taking the value 1 with probability $\frac{d}{|V| - 1}$. Thus

$$|N_{in}(S_1) \setminus S_1| = \sum_{v \in V \setminus S_1} \mathbb{I}[v \in N_{in}(S_1)]$$

is a sum of $|V| - 1$ independent Bernoulli random variables. Thus $|N_{in}(S_1) \setminus S_1|$ obeys the Binomial distribution $\text{Bin}(|V| - 1, \frac{d}{|V| - 1})$ which has mean $d$ and variance $d(1 - \frac{d}{|V| - 1})$. Now let $W = |N_{in}(S_1) \setminus S_1|$. Then by Chebyshev’s inequality for $t > 0$

$$P(|W - d| \geq td) \leq \frac{(1 - \frac{d}{|V| - 1})}{td^2} \leq \frac{1}{td^2}.$$ 

We note since $d \geq 2$ we can choose $t = \frac{\sqrt{3}}{d}$ so that $\frac{1}{td^2} \leq \frac{3}{4}$. Thus with probability at least $1/4$ we have that

$$|N_{in}(S_1) \setminus S_1| \in [(1 - t)d, (1 + t)d].$$

It follows that $\mathbb{E}[Z] = \Omega(|S_1|d^2)$. 

\[\square\]

**D. Stochastic forgetting**

Below we provide a general definition for probabilistic un-learning:

**Definition 1.** $(\alpha, \beta)$-sharded-unlearning: Consider an algorithm $\mathcal{A}$ that, given a shard graph $G$ as input, outputs a model $\mathcal{A}(G)$ trained on the shards of $G$. If $G'$ is a shard
Figure 9. Shard-Graph in SAFE-DP: In SAFE-DP each shard when training adapters uses its own data non-privately, but enforces DP when using data from other shards. This can be visualized as shard graphs with weighted edges, where the edge weight measures the privacy leakage between the interacting nodes.

**Theorem 2.** [16] Group privacy: Let \( A \) be an \((\epsilon, \delta)\)-DP algorithm. Then for all \( D, D' \), such that \( D \) and \( D' \) differ in at most \( k \) samples, we get the following result:

\[
P(A(D) \in \Theta) \leq e^\epsilon P(A(D') \in \Theta) + \delta_g,
\]

where \( \epsilon_g = k\epsilon \) and \( \delta_g = \frac{e^{k\epsilon} - 1}{e^\epsilon - 1} \delta \).

We observe that the group privacy result, i.e. \((\epsilon_g, \delta_g) = (e^{k\epsilon}, \frac{e^{k\epsilon} - 1}{e^\epsilon - 1} \delta)\), weakens with increasing \( k \). This necessitates the notion of a privacy budget \((\alpha, \delta_0)\) which is an upper bound for the privacy leakage accumulation with successive forgetting requests. This budget can be chosen by the user, or in some cases, constrained by the fact that \( \frac{e^{k\epsilon} - 1}{e^\epsilon - 1} \delta \leq 1 \).

Differential privacy can detrimentally reduce the utility (accuracy) of a model. Hence, blindly using DP-SGD (or any another DP method) during training may result in models that, while strongly private (and hence not requiring frequent re-training) attain significantly lower accuracy. SAFE provides a simple yet effective unlearning mechanism which enables forgetting shards, by re-training the neighbouring contaminated shards (requires frequent yet reduced re-training, but higher accuracy). These observations inspire us to design an algorithm which combines SAFE and Stochastic Forgetting:

**SAFE-DP:** For each shard \( S_i \) in the shard graph \( G \), we train a binary classification model which treats the data at node \( S_i \) as the positive class (without privacy), and data at neighbouring nodes pointing to it as the negative classes (with privacy). When asked to forget a sample in \( S_i \) we can simply drop the classifier for the shard \( S_i \) (later re-train if mandated by the privacy cost of the budget), without having to worry about re-training models corresponding to other shards which used \( S_i \) as a negative class (since the data of the negative shards was trained with DP).

Unlike the standard version of SAFE, where all the connections of shard \( S_i \) (using \( S_i \) for training) need to be re-trained upon a single forget request, SAFE-DP allows us to keep the remaining neighboring shards while accounting for some cost to the privacy budget. By allowing each shard \( S_i \) to use its own data non-privately, SAFE-DP also provides better utility compared to completely private models (trained with DP). While training with SAFE-DP, we compute the total \((\alpha, \beta)\) using the composition property of DP [16, 37, 26].

**Privacy Accounting:** The SAFE-DP algorithm is defined with an accompanying privacy level given by \((\epsilon, \delta)\), which sets the DP training parameters in SAFE-DP (using DP-SGD [1, 24]). While the \((\alpha, \beta)\) parameters in the unlearning definition (Definition 1) are dependent on the number of...
sequential forgetting requests \( k \), SAFE-DP provides the following unlearning guarantees: \( \alpha = k \epsilon \) and \( \beta = \frac{e^{\kappa \epsilon} - 1}{e^{\kappa \epsilon} - 1} \delta \), where \( \beta \leq 1 \) (from Theorem 2).

Before the start of training, the user chooses a desired forgetting budget \((\alpha_b, \beta_b)\) (with \( \beta_b \leq 1 \)), which measures the information contained (privacy leakage) about the user after forgetting her samples. The forgetting algorithm should respect the budget while providing non-vacuous guarantees. This provides us with the following bound for the number of unlearning request before full re-training: When \( \delta \ll 1 \) and \( \delta \ll \beta_b \), we can approximate this result with

\[
k = \min \left( \frac{\alpha_b}{\epsilon}, \frac{\log (\beta_b(e^\epsilon - 1)/\delta + 1)}{\epsilon} \right).
\]

For our experiments we assume \( \alpha_b = 30 \) and \( \beta_b = 1 \). We vary \( \epsilon = \{1, 2, 3, 4\} \) and \( \delta = 1e^{-10}, 1e^{-11}, 1e^{-12}, 1e^{-13} \) and choose different values for \( k \). Note that we can choose the values for \( k \) by varying both \( \epsilon \) and \( \delta \) (ensuring \( \delta \ll 1 \)) and choosing the best value by comparing the accuracy on a held out validation set.

E. Additional Details of Architecture and Training

Optimized graph structures. For our SAFE method, in Section 5 we described the bi-level sharding structure where we split the dataset into a number \( n_c \) “coarse shards” via class balanced sub-sampling which are further split into \( n_f \) “fine shards” via class partitioning. The total number of shards is given by \( n = n_c \cdot n_f \). In Table 4 we report the choice of \((n_c, n_f)\) used for each dataset for each value of \( n \) for the SAFE method.

Optimization. Whenever training InCA adapters, we train using AdamW for 30 epochs using cosine annealing and starting learning rate \( \text{lr} = 0.05 \); we use weight decay of \( 10^{-4} \). Whenever training the linear model for the experiment in Fig. 7 we lower the learning rate to \( \text{lr} = 3e^{-4} \) as we observed this increased performance. For the linear model all other hyperparameters remain the same as when training the InCA adapter.

Architecture. When using InCA adapters, LayerNorm is applied to both the inputs and queries separately before they are passed through the cross-attention block. A second LayerNorm is applied after the cross-attention before the final logits are computed as customary in ViTs. While in general InCA adapters can be applied to any layer in the network [15], in our experiments we always attach it to the penultimate layer, namely the end of block 22 (the input to blocks.23.norm1). When using linear adapters for the experiment depicted in Fig. 7, we also apply LayerNorm before the fully connected layer.

Prototype model. We note that the prototype model is only added at inference time, not training time, as adding the prototype model during training would expose each adapter to information from all other training samples, thus breaking the information compartmentalization. When computing the prototypes \( p_k \), we normalize the feature embeddings

\[
p_k = \frac{1}{N_c} \sum_{(x,y) \in D(k)} \frac{f_w(x)}{\|f_w(x)\|},
\]

which makes the prototypes less sensitive to outliers and leads to better predictions.

Dataset details. In Table 3 we report the size of the training and testing splits and number of classes for the 7 datasets we consider, and links to access the data.

F. Additional Related Work

In our work we present SAFE as a method that flexibly learns a model composed of model parts that are learned using different data (as prescribed by the Shard Graph). This heterogeneous routing of data during training enables training the different model components in parallel and permits training hundreds of models quickly and efficiently. While we apply this approach for the problem of forgetting, recently “heterogeneous data routing” has also been the focus of much work in enabling better massive model scaling. In these works, “heterogeneous data routing” is used to route different data and activations to different model parts and distributing the computation to more computing nodes. Through distribution of computation, one can reduce the inference and training costs of foundation models and enable even more parameters than what is permissible by a single monolithic model [17]. In the work of [17] a large language model based on the transformer architecture is built with
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Training Images</th>
<th>Testing Images</th>
<th># Classes</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltech-256</td>
<td>15,418</td>
<td>15,189</td>
<td>257</td>
<td><a href="https://authors.library.caltech.edu/7694/">https://authors.library.caltech.edu/7694/</a></td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>50,000</td>
<td>10,000</td>
<td>100</td>
<td><a href="https://www.cs.toronto.edu/~kriz/cifar.html">https://www.cs.toronto.edu/~kriz/cifar.html</a></td>
</tr>
<tr>
<td>CUB-200</td>
<td>5,994</td>
<td>5,794</td>
<td>200</td>
<td><a href="https://www.vision.caltech.edu/datasets/cub_200_2011/">https://www.vision.caltech.edu/datasets/cub_200_2011/</a></td>
</tr>
<tr>
<td>DTD</td>
<td>4,230</td>
<td>1,410</td>
<td>47</td>
<td><a href="https://www.robots.ox.ac.uk/~vgg/data/dtd/">https://www.robots.ox.ac.uk/~vgg/data/dtd/</a></td>
</tr>
<tr>
<td>Stanford Cars</td>
<td>8,144</td>
<td>8,041</td>
<td>196</td>
<td><a href="https://ai.stanford.edu/~jkrause/cars/car_dataset.html">https://ai.stanford.edu/~jkrause/cars/car_dataset.html</a></td>
</tr>
<tr>
<td>Stanford Dogs</td>
<td>12,000</td>
<td>8,580</td>
<td>120</td>
<td><a href="http://vision.stanford.edu/aditya86/imagenetdogs/">http://vision.stanford.edu/aditya86/imagenetdogs/</a></td>
</tr>
</tbody>
</table>

Table 3. Dataset Information. We report the number of classes as well as the number of training and testing images for each dataset, as well as links to download the datasets.

dynamic execution layers, where the model’s intermediate activations are routed into disjoint layers based on their representations via “switching layers”. This is generalized in the work of Pathways [3] that creates the necessary infrastructure to train such models on distributed computing systems and allows training state-of-the-art language models [12]. The latest developments [18, 19] of this method apply heterogeneous propagation of data in connection with multi-task learning where “agent networks” cooperate with partner-agent representations to adapt and solve new tasks. We note while both [17, 3] and our work utilize heterogeneous data routing, in our work data routing and compartmentalization is deterministic and is based on pre-specified data usage rules codified by the Shard Graph, as opposed to selecting the data routing based on the data’s representations. Overall our work focuses on the problem of forgetting rather than multi-task learning and increasing the model scale.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>No sharding</th>
<th>Prototypes</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltech-256</td>
<td>94.3%</td>
<td>93.2%</td>
<td>94.2%</td>
<td>94.1%</td>
<td>94.0%</td>
<td>93.7%</td>
<td>93.7%</td>
<td>93.5%</td>
<td>93.2%</td>
<td>93.3%</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>83.1%</td>
<td>71.2%</td>
<td>84.4%</td>
<td>84.6%</td>
<td>84.1%</td>
<td>83.3%</td>
<td>82.8%</td>
<td>82.2%</td>
<td>81.5%</td>
<td>80.9%</td>
</tr>
<tr>
<td>CUB-200</td>
<td>88.3%</td>
<td>85.8%</td>
<td>88.6%</td>
<td>87.9%</td>
<td>86.1%</td>
<td>85.8%</td>
<td>85.3%</td>
<td>85.5%</td>
<td>82.5%</td>
<td>84.5%</td>
</tr>
<tr>
<td>DTD</td>
<td>77.8%</td>
<td>73.8%</td>
<td>78.3%</td>
<td>78.3%</td>
<td>77.1%</td>
<td>75.4%</td>
<td>75.5%</td>
<td>75.1%</td>
<td>73.9%</td>
<td>74.2%</td>
</tr>
<tr>
<td>MIT-67</td>
<td>87.9%</td>
<td>85.8%</td>
<td>88.1%</td>
<td>87.7%</td>
<td>86.9%</td>
<td>86.3%</td>
<td>86.5%</td>
<td>86.0%</td>
<td>86.2%</td>
<td>86.4%</td>
</tr>
<tr>
<td>Stanf. Cars</td>
<td>75.7%</td>
<td>41.0%</td>
<td>72.6%</td>
<td>68.5%</td>
<td>62.1%</td>
<td>58.3%</td>
<td>53.2%</td>
<td>47.4%</td>
<td>41.9%</td>
<td>36.2%</td>
</tr>
<tr>
<td>Stanf. Dogs</td>
<td>87.9%</td>
<td>88.0%</td>
<td>88.9%</td>
<td>89.2%</td>
<td>89.2%</td>
<td>89.0%</td>
<td>88.6%</td>
<td>88.3%</td>
<td>87.7%</td>
<td>87.8%</td>
</tr>
</tbody>
</table>

Avg. | 85.0% | 77.0% | 85.0% | 84.3% | 82.8% | 81.7% | 80.8% | 79.4% | 78.1% | 77.6% |

Table 4. **Coarse vs. fine shard split.** We report the number of coarse and fine shards, \((n_c, n_f)\), used for each dataset at the different sharding levels.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>No sharding</th>
<th>Prototypes</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltech-256</td>
<td>94.3%</td>
<td>93.2%</td>
<td>94.2%</td>
<td>94.1%</td>
<td>93.6%</td>
<td>92.8%</td>
<td>92.8%</td>
<td>90.1%</td>
<td>86.7%</td>
<td>84.9%</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>83.1%</td>
<td>71.2%</td>
<td>84.5%</td>
<td>84.6%</td>
<td>84.0%</td>
<td>83.2%</td>
<td>82.0%</td>
<td>80.7%</td>
<td>79.7%</td>
<td>77.6%</td>
</tr>
<tr>
<td>CUB-200</td>
<td>88.3%</td>
<td>85.8%</td>
<td>88.5%</td>
<td>87.6%</td>
<td>83.9%</td>
<td>73.1%</td>
<td>65.3%</td>
<td>65.1%</td>
<td>62.8%</td>
<td>63.7%</td>
</tr>
<tr>
<td>DTD</td>
<td>77.8%</td>
<td>73.8%</td>
<td>78.3%</td>
<td>78.1%</td>
<td>76.1%</td>
<td>74.6%</td>
<td>71.8%</td>
<td>65.1%</td>
<td>58.6%</td>
<td>52.6%</td>
</tr>
<tr>
<td>MIT-67</td>
<td>87.9%</td>
<td>85.8%</td>
<td>88.1%</td>
<td>87.7%</td>
<td>86.8%</td>
<td>85.2%</td>
<td>83.6%</td>
<td>81.5%</td>
<td>77.5%</td>
<td>77.9%</td>
</tr>
<tr>
<td>Stanf. Cars</td>
<td>75.7%</td>
<td>41.0%</td>
<td>72.1%</td>
<td>66.0%</td>
<td>55.8%</td>
<td>40.9%</td>
<td>23.7%</td>
<td>17.1%</td>
<td>13.2%</td>
<td>7.0%</td>
</tr>
<tr>
<td>Stanf. Dogs</td>
<td>87.9%</td>
<td>88.0%</td>
<td>88.8%</td>
<td>89.2%</td>
<td>89.2%</td>
<td>89.0%</td>
<td>88.6%</td>
<td>88.3%</td>
<td>87.7%</td>
<td>87.8%</td>
</tr>
</tbody>
</table>

Avg. | 85.0% | 77.0% | 85.0% | 84.3% | 82.8% | 81.7% | 80.8% | 79.4% | 78.1% | 77.6% |

Table 5. **SAFE Accuracy at different sharding scales.** We report the accuracy of SAFE across different sharding scales.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>No sharding</th>
<th>Prototypes</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltech-256</td>
<td>94.3%</td>
<td>93.2%</td>
<td>94.2%</td>
<td>94.1%</td>
<td>93.6%</td>
<td>92.8%</td>
<td>90.1%</td>
<td>86.8%</td>
<td>86.4%</td>
<td>90.5%</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>83.1%</td>
<td>71.2%</td>
<td>84.5%</td>
<td>84.6%</td>
<td>84.0%</td>
<td>83.2%</td>
<td>82.0%</td>
<td>80.7%</td>
<td>79.7%</td>
<td>77.7%</td>
</tr>
<tr>
<td>CUB-200</td>
<td>88.3%</td>
<td>85.8%</td>
<td>88.5%</td>
<td>87.6%</td>
<td>83.9%</td>
<td>73.1%</td>
<td>65.3%</td>
<td>65.1%</td>
<td>62.8%</td>
<td>63.7%</td>
</tr>
<tr>
<td>DTD</td>
<td>77.8%</td>
<td>73.8%</td>
<td>78.3%</td>
<td>78.1%</td>
<td>76.1%</td>
<td>74.6%</td>
<td>71.8%</td>
<td>65.1%</td>
<td>58.6%</td>
<td>52.6%</td>
</tr>
<tr>
<td>MIT-67</td>
<td>87.9%</td>
<td>85.8%</td>
<td>88.1%</td>
<td>87.7%</td>
<td>86.8%</td>
<td>85.2%</td>
<td>83.6%</td>
<td>81.5%</td>
<td>77.5%</td>
<td>77.9%</td>
</tr>
<tr>
<td>Stanf. Cars</td>
<td>75.7%</td>
<td>41.0%</td>
<td>72.1%</td>
<td>66.0%</td>
<td>55.8%</td>
<td>40.9%</td>
<td>23.7%</td>
<td>17.1%</td>
<td>13.2%</td>
<td>7.0%</td>
</tr>
<tr>
<td>Stanf. Dogs</td>
<td>87.9%</td>
<td>88.0%</td>
<td>88.8%</td>
<td>89.2%</td>
<td>89.2%</td>
<td>89.0%</td>
<td>88.6%</td>
<td>88.3%</td>
<td>87.7%</td>
<td>87.8%</td>
</tr>
</tbody>
</table>

Avg. | 85.0% | 77.0% | 84.9% | 83.9% | 81.4% | 76.9% | 71.9% | 68.5% | 65.4% | 63.3% |

Table 6. **SISA Accuracy at different sharding scales.** We report the accuracy of SISA across different sharding scales.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>No sharding</th>
<th>Prototypes</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltech-256</td>
<td>94.3%</td>
<td>93.2%</td>
<td>94.2%</td>
<td>94.1%</td>
<td>93.6%</td>
<td>92.8%</td>
<td>90.1%</td>
<td>86.8%</td>
<td>86.4%</td>
<td>90.5%</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>83.1%</td>
<td>71.2%</td>
<td>84.5%</td>
<td>84.6%</td>
<td>84.0%</td>
<td>83.2%</td>
<td>82.0%</td>
<td>80.7%</td>
<td>79.7%</td>
<td>77.7%</td>
</tr>
<tr>
<td>CUB-200</td>
<td>88.3%</td>
<td>85.8%</td>
<td>88.5%</td>
<td>87.6%</td>
<td>83.9%</td>
<td>73.1%</td>
<td>65.3%</td>
<td>65.1%</td>
<td>62.8%</td>
<td>63.7%</td>
</tr>
<tr>
<td>DTD</td>
<td>77.8%</td>
<td>73.8%</td>
<td>78.3%</td>
<td>78.1%</td>
<td>76.1%</td>
<td>74.6%</td>
<td>71.8%</td>
<td>65.1%</td>
<td>58.6%</td>
<td>52.6%</td>
</tr>
<tr>
<td>MIT-67</td>
<td>87.9%</td>
<td>85.8%</td>
<td>88.1%</td>
<td>87.7%</td>
<td>86.8%</td>
<td>85.2%</td>
<td>83.6%</td>
<td>81.5%</td>
<td>77.5%</td>
<td>77.9%</td>
</tr>
<tr>
<td>Stanf. Cars</td>
<td>75.7%</td>
<td>41.0%</td>
<td>72.1%</td>
<td>66.0%</td>
<td>55.8%</td>
<td>40.9%</td>
<td>23.7%</td>
<td>17.1%</td>
<td>13.2%</td>
<td>7.0%</td>
</tr>
<tr>
<td>Stanf. Dogs</td>
<td>87.9%</td>
<td>88.0%</td>
<td>88.8%</td>
<td>89.2%</td>
<td>89.2%</td>
<td>89.0%</td>
<td>88.6%</td>
<td>88.3%</td>
<td>87.7%</td>
<td>87.8%</td>
</tr>
</tbody>
</table>

Avg. | 85.0% | 77.0% | 84.9% | 83.9% | 81.4% | 76.9% | 71.9% | 68.5% | 65.4% | 63.3% |

Table 7. **ProtoSISA Accuracy at different sharding scales.** We report the accuracy of ProtoSISA across different sharding scales.