Ticketed Learning–Unlearning Schemes

Badih Ghazi BADIHGHAZI@GMAIL.COM
Pritish Kamath PRITISH@ALUM.MIT.EDU
Ravi Kumar RAVI.K53@GMAIL.COM
Google Research, Mountain View

Pasin Manurangsi PASIN@GOOGLE.COM
Google Research, Thailand

Ayush Sekhari SEKHARI@MIT.EDU
Massachusetts Institute of Technology

Chiyuan Zhang CHIYUAN@GOOGLE.COM
Google Research, Mountain View

Editors: Gergely Neu and Lorenzo Rosasco

Abstract

We consider the *learning–unlearning* paradigm defined as follows. First given a dataset, the goal is to learn a good predictor, such as one minimizing a certain loss. Subsequently, given any subset of examples that wish to be *unlearnt*, the goal is to learn, without the knowledge of the original training dataset, a good predictor that is identical to the predictor that would have been produced when learning from scratch on the surviving examples.

We propose a new *ticketed* model for learning–unlearning wherein the learning algorithm can send back additional information in the form of a small-sized (encrypted) "ticket" to each participating training example, in addition to retaining a small amount of "central" information for later. Subsequently, the examples that wish to be unlearnt present their tickets to the unlearning algorithm, which additionally uses the central information to return a new predictor. We provide space-efficient ticketed learning–unlearning schemes for a broad family of concept classes, including thresholds, parities, intersection-closed classes, among others.

En route, we introduce the *count-to-zero* problem, where during unlearning, the goal is to simply know if there are any examples that survived. We give a ticketed learning–unlearning scheme for this problem that relies on the construction of Sperner families with certain properties, which might be of independent interest.

Keywords: Machine unlearning, data deletion, ticket model, space complexity

1. Introduction

Machine learning models trained on user data have become widespread in applications. While these models have proved to be greatly valuable, there is an increasing demand for ensuring that they respect the consent of the users and the privacy of their data. One of the simplest and common challenge is how to update a model when a user wishes to drop out of the training data. Re-learning the model from scratch without this user's data is a natural approach, but this can be computationally prohibitive. Designing alternative approaches which aim to "mimic" this natural approach, without the computational overhead, has come to be known as the problem of machine "unlearning", a topic of growing interest.

Informally, the ideal *learning–unlearning (LU)* paradigm can be modeled as follows. An agent gets a sequence of learning and unlearning requests. Each such request is accompanied by a dataset

© 2023 B. Ghazi, P. Kamath, R. Kumar, P. Manurangsi, A. Sekhari & C. Zhang.

of examples. At any point, the agent must be able to produce a predictor, the distribution of which must be identical to, or at least "close to", that of the same agent on a single learning request containing only the *surviving* examples, namely, all examples in learning requests that have not been present in any subsequent unlearning request. A naive approach to keep in mind is an agent that explicitly keeps track of all surviving examples at any point, and returns the predictor obtained by simulating a single learning request on the surviving examples. However, the space requirement of such an agent is linear in the number of examples (as it needs to store all the examples to simulate re-training). The goal in this paper is to understand:

*When are **space-efficient** learning–unlearning schemes possible?*

Unlearning has been an active area of research. Cao and Yang (2015) initiated the study through *exact* unlearning. Their definition requires an algorithm to have identical outputs on a dataset after deleting a point, as if that point was never inserted; their algorithms are restricted to very structured problems. Several later works studied unlearning algorithms for empirical risk minimization (Guo et al., 2020; Izzo et al., 2021; Neel et al., 2021; Ullah et al., 2021; Thudi et al., 2022; Graves et al., 2021; Bourtoule et al., 2021). However, these works focus exclusively on the *time complexity* of unlearning. However, many of these provably effective unlearning methods have large space requirements to enable deletion, e.g., they store space-intensive check-pointing data structures (Ullah et al., 2021), large ensembles of models trained on subsets of data (Bourtoule et al., 2021), extensive statistics about the learning process (Thudi et al., 2022; Neel et al., 2021), or at the very least the entire (surviving) training dataset. This additional space overhead can be impractical for various applications. In contrast, the primary focus of our paper is to understand the *space complexity* of unlearning and to develop space-efficient learning–unlearning schemes for general function classes.

The model of learning–unlearning that has been considered in the above mentioned prior works can be deemed as the "central" model, where the agent responsible for unlearning has to remember all additional information beyond the learnt predictor that might be required for unlearning later.

1.1. Our Contributions

In Section 2, we introduce the notion of a *ticketed learning–unlearning (TiLU) scheme*, which consists of a *learning* and an *unlearning* algorithm. The learning algorithm given a training dataset, produces a predictor, as well as a "ticket" corresponding to each example and some additional "central" information. The unlearning algorithm, given a subset of the training examples accompanied by their corresponding tickets, and the previously generated central information, produces an updated predictor (that realizes the unlearning guarantee).

In Section 3, we show that certain limitations of the central model of learning–unlearning can be overcome by the ticketed model. In particular, we provide TiLU schemes for a broad family of concept classes that includes several commonly studied classes such as one-dimensional thresholds, product of thresholds, and parities. In Section 4, we provide improved TiLU schemes with even better space complexity bounds for products of thresholds, as well as for point functions, which are notably not covered by the techniques in Section 3.

Underlying our improvements in Section 4, is a basic primitive of *count-to-zero*. The goal in this problem is simply to determine if the unlearning request contains precisely all the examples in the original learning request or not. We give a TiLU scheme for this problem with space complexity that scales as the log of the inverse-Ackermann function (see Definition 9) of the number of examples.

This relies on a novel construction of *size-indexing Sperner families*, which we believe to be of independent interest. We also prove a lower bound on such families in Section 5, and use it to show that the space complexity of TiLU schemes for any non-trivial concept class must necessarily increase with the number of examples.

1.2. Other Related Work

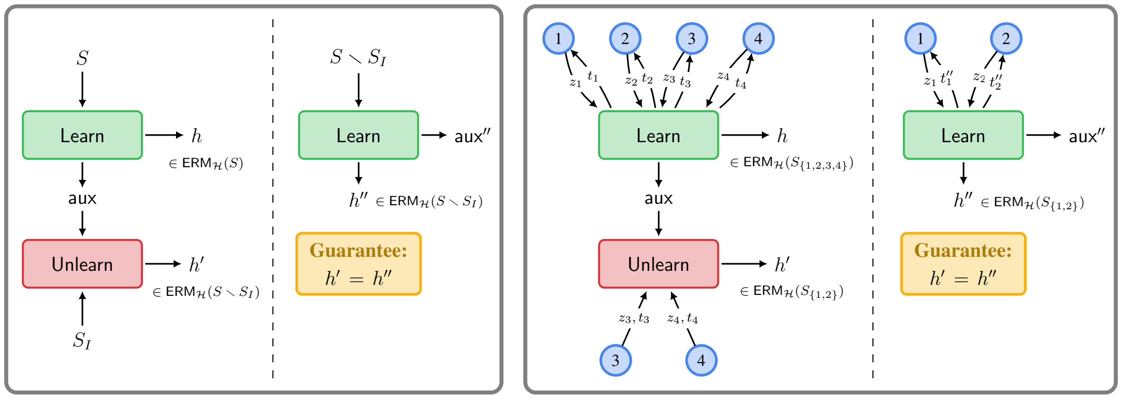
For specific learning models like SVMs, various algorithms for exact unlearning have been considered under the framework of decremental learning (Cauwenberghs and Poggio, 2000; Tveit et al., 2003; Karasuyama and Takeuchi, 2010; Romero et al., 2007). However, these works do not enjoy any guarantees on the space requirements for unlearning. The primary motivation in these works is to use the decremental learning framework to empirically estimate the leave-one-out error in order to provide generalization guarantees for the learned model.

We also note that beyond exact unlearning, various probabilistic/approximate notions of unlearning, which are in turn inspired by the notions in differential privacy (Dwork et al., 2006), have been considered (Ginart et al., 2019; Sekhari et al., 2021; Gupta et al., 2021; Chourasia et al., 2023), and there has been an extensive effort to develop (approximate) unlearning algorithms for various problem settings. These include unlearning in deep neural networks (Du et al., 2019; Golatkar et al., 2020, 2021; Nguyen et al., 2020; Graves et al., 2021), random forests (Brophy and Lowd, 2021), large-scale language models (Zanella-Béguelin et al., 2020), convex loss functions (Guo et al., 2020; Sekhari et al., 2021; Neel et al., 2021; Suriyakumar and Wilson, 2022), etc. However, we reiterate that all these prior works have huge space requirements, especially in high-dimensional settings.

In a related setting, prior works have looked at the problem of constructing *history-independent* data structures (Hartline et al., 2005; Naor and Teague, 2001). The key motivation here is to prevent an adversary from inferring information about the dataset from the memory representation of the data structure that is not available through its "legitimate" interface. However, no direct application of history-independent data structures for unlearning is currently known.

The main motivation for unlearning is that a trained model could potentially leak user information in various forms such as membership inference attack (Shokri et al., 2017; Carlini et al., 2022a) or even data extraction attack for text (Carlini et al., 2021, 2023b) or image (Carlini et al., 2023a) generative models. An unlearning protocol allows users to withdraw their data from the model training set. The implication of such mechanisms were also studied in the literature. For example, Ippolito et al. (2022) showed that an inference-time filtering mechanism to prevent the generation of verbatim training text sequences could be easily bypassed by "style-transfer" like prompting in large language models. Carlini et al. (2022b) further show a "privacy onion" effect where unlearning one set of examples could have large impact on the privacy score of a non-overlapping set of examples in the training set. Furthermore, there have also been works exploring machine unlearning as a tool to attack an ML system, e.g., Di et al. (2022) recently showed that unlearning can be used as a trigger for executing data poisoning attacks (on demand).

Finally, there has been a recent line of work on formulating alternative definitions of approximate (and probabilistic) unlearning to capture the spirit of "right to be forgotten" and "data deletion" under different circumstances, e.g., (Dukler et al., 2023; Krishna et al., 2023; Cohen et al., 2022; Eisenhofer et al., 2022; Garg et al., 2020), etc. However, developing space-efficient algorithms for these definitions is beyond the scope of our paper, and we only consider exact unlearning of ERMs.


S
Learn
h
$\in \mathsf{ERM}_{\mathcal{H}}(S)$
aux
Unlearn
h'
$\in \mathsf{ERM}_{\mathcal{H}}(S \setminus S_I)$
S_I
$S \setminus S_I$
Learn
aux$''$
$h'' \in \mathsf{ERM}_{\mathcal{H}}(S \setminus S_I)$
Guarantee:
$h' = h''$
1
2
3
4
z_1 t_1
z_2 t_2
z_3 t_3
z_4 t_4
Learn
h
$\in \mathsf{ERM}_{\mathcal{H}}(S_{\{1,2,3,4\}})$
aux
Unlearn
h'
$\in \mathsf{ERM}_{\mathcal{H}}(S_{\{1,2\}})$
z_3, t_3
z_4, t_4
3
4
1
2
z_1 t_1''
z_2 t_2''
Learn
aux$''$
h'' $\in \mathsf{ERM}_{\mathcal{H}}(S_{\{1,2\}})$
Guarantee:
$h' = h''$

(a) LU Scheme (Definition 1)   (b) TiLU Scheme (Definition 2)

Figure 1: Illustration of the guarantees of LU schemes in central and ticketed models.

2. Learning–Unlearning Schemes

We focus on supervised learning with the binary loss function $\ell(\hat{y}, y) = \mathbb{1}\{\hat{y} \neq y\}$. Each example (x, y) belongs to $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, with the label set $\mathcal{Y} = \{0, 1\}$. We denote the *empirical loss* of a predictor $h : \mathcal{X} \to \mathcal{Y}$ on a dataset $S = (z_1 = (x_1, y_1), \ldots, z_n = (x_n, y_n)) \in \mathcal{Z}^n$ as[1] $\mathcal{L}(h; S) := \sum_{i \in [n]} \ell(h(x_i), y_i)$. For any concept class $\mathcal{H} \subseteq (\mathcal{X} \to \mathcal{Y})$, we say that a dataset S is *$\mathcal{H}$-realizable* if there exists $h \in \mathcal{H}$ such that $\mathcal{L}(h; S) = 0$. Let $\mathsf{ERM}_{\mathcal{H}}(S) = \operatorname{argmin}_{h \in \mathcal{H}} \mathcal{L}(h; S)$ be the set of all minimizers in $\mathcal{H}$ of the empirical loss over S. For any subset $I \subseteq [n]$ of indices, let S_I denote the dataset $((x_i, y_i))_{i \in I}$.

Central Model of Learning–Unlearning. We define the notion of a *learning–unlearning (LU) scheme* below, formalizing the standard setting considered in literature, which we will refer to as the "central" model.

Definition 1 (LU Scheme) *A* learning–unlearning (LU) scheme *for a concept class $\mathcal{H}$ consists of a pair* $(\mathsf{Learn}, \mathsf{Unlearn})$ *of algorithms as follows.*

- *On input $S \in \mathcal{Z}^n$, $\mathsf{Learn}(S)$ returns a predictor $h \in \mathsf{ERM}_{\mathcal{H}}(S)$ and auxiliary information $\mathsf{aux} \in \{0,1\}^C$.*
- *For $I \subseteq [n]$, $\mathsf{Unlearn}(S_I, \mathsf{aux})$ returns a predictor $h' \in \mathsf{ERM}_{\mathcal{H}}(S \setminus S_I)$.*

For all S and $S_I \subseteq S$, the predictor returned by $\mathsf{Unlearn}$ is required to be identical to the predictor returned by $\mathsf{Learn}(S \setminus S_I)$. The space complexity of the scheme is C, the bit-complexity of the auxiliary information aux.

Unless otherwise stated, we will only require the above to hold for $\mathcal{H}$-realizable S.

Informally speaking, in a LU scheme, the central agent on a learning request containing a dataset, returns a predictor and also retains some auxiliary information for itself. On an unlearning request (containing a subset of the previous dataset), the agent, using its auxiliary information, returns a new predictor. This new predictor is required to be identical to the predictor that the agent would have returned on a learning request with only the surviving examples. See Figure 1(a).

1. For ease of notation, we let $\mathcal{L}$ denote the sum of losses over the dataset, as opposed to the more conventional average.

In this work, our focus is on *space complexity*, namely, we consider a LU scheme as *space-efficient* if its space complexity is $\mathrm{poly}(\log n, \log |\mathcal{Z}|, \log |\mathcal{H}|)$. Note that the naive scheme, where aux contains the entire dataset S and $\mathsf{Unlearn}$ simply returns the predictor returned by $\mathsf{Learn}(S \smallsetminus S_I)$, requires $C = n \cdot \log |\mathcal{Z}|$ bits and hence is not space-efficient. While we do not explicitly discuss time complexity, all the algorithms that we present are also time-efficient, namely, they run in $\mathrm{poly}(n, \log |\mathcal{Z}|, \log |\mathcal{H}|)$ time. Our lower bounds, on the other hand, hold even against computationally inefficient algorithms. Finally, in this work, we only assume a *one-shot* setting, i.e., there is a single learning request with a dataset S, followed by a single unlearning request $S_I \subseteq S$.

We show in Appendix B that there exists a space-efficient LU scheme for the class of threshold functions. Unfortunately, the central model easily runs into barriers and is quite limited: we also show that any LU scheme for the class of point functions must store $\Omega(|\mathcal{X}|)$ bits of auxiliary information. To circumvent such barriers, we introduce a new *ticketed* model of learning–unlearning.

Ticketed Model of Learning–Unlearning. The basic idea of the ticketed model is that the central agent issues "tickets" for each example that is stored by the corresponding user contributing the example. These tickets have to then be provided as part of the unlearning request (see Figure 1(b)).

Definition 2 (Ticketed LU Scheme) *A* ticketed learning–unlearning (TiLU) scheme *for a concept class $\mathcal{H}$ consists of a pair* $(\mathsf{Learn}, \mathsf{Unlearn})$ *of algorithms such that*

- *On input $S = (z_1, \ldots, z_n) \in \mathcal{Z}^n$,* $\mathsf{Learn}(S)$ *returns $(h, \mathsf{aux}, (t_i)_{i \in [n]})$, with a predictor $h \in \mathsf{ERM}_{\mathcal{H}}(S)$, auxiliary information* $\mathsf{aux} \in \{0,1\}^{C_s}$ *and* tickets $t_1, \ldots, t_n \in \{0,1\}^{C_t}$ *associated with examples $z_1, \ldots, z_n$ respectively*[2].
- *On input $I \subseteq [n]$,* $\mathsf{Unlearn}(S_I, \mathsf{aux}, (t_i)_{i \in I})$ *returns a predictor $h' \in \mathsf{ERM}_{\mathcal{H}}(S \smallsetminus S_I)$.*

For all S and $S_I \subseteq S$, the predictor returned by $\mathsf{Unlearn}$ *above is required to be identical to the predictor returned by* $\mathsf{Learn}(S \smallsetminus S_I)$*. The space complexity of the scheme is (C_s, C_t), where C_s is the bit-complexity of* aux *and C_t is the bit-complexity of each ticket t_i.*

Unless otherwise stated, we will only require the above to hold for $\mathcal{H}$-realizable S.

A TiLU scheme is similar to the central LU scheme, except that during learning, the central agent issues a "ticket" for each example in the dataset that will be given to the user contributing that said example; this ticket is required along with the example as part of the unlearning request. As before, we consider a TiLU scheme to be *space-efficient* if $C_s, C_t = \mathrm{poly}(\log n, \log |\mathcal{Z}|, \log |\mathcal{H}|)$. Note that while the space requirement for storing all tickets does grow linearly in n, the main point is that no single party has to store more than poly-logarithmic amount of information. The challenge in constructing a TiLU scheme is that only the tickets corresponding to the examples in the unlearning request are available at the time of unlearning, so the unlearning step has to be performed with access to a limited amount of information.

Remark 3 For both schemes, our definition is restrictive in the following sense:

- *Exact ERM*: the learning algorithm is required to output a predictor in $\mathsf{ERM}_{\mathcal{H}}(S)$.
- *Exact unlearning*: the predictor after unlearning is required to be identical to the predictor learned on just the surviving examples.[3]
- *One-shot unlearning:* there is a single learning request, followed by a single unlearning request.

2. The tickets are sent back to the users contributing the corresponding examples, and not stored centrally.
3. We could consider a seemingly more relaxed variant where the *distribution* of the predictor returned by Unlearn equals the distribution of the predictor returned by Learn on the surviving examples, but given any such scheme, we could convert it to have a deterministic output by simply choosing a canonical predictor for each distribution.

While the above restrictions are seemingly limiting, as our results show, they already capture and highlight many of the key technical challenges. Developing schemes when these restrictions are suitably relaxed (e.g., when unlearning can be approximate) is an important research direction.

3. Mergeable Concept Classes

We define a *mergeability* property of concept classes and provide examples of several commonly studied classes with this property. We then provide space-efficient TiLU schemes for such classes.

Definition 4 (Mergeable Concept Class) *A concept class $\mathcal{H} \subseteq (\mathcal{X} \to \mathcal{Y})$ is said to be C-*bit mergeable *if there exist methods* Encode, Merge, Decode *such that*

- $\mathsf{Encode} : \mathcal{Z}^* \to \{0,1\}^C$ *is a permutation-invariant encoding of its input into C bits.*
- $\mathsf{Decode} : \{0,1\}^C \to \mathcal{H}$ *such that* $\mathsf{Decode}(\mathsf{Encode}(S)) \in \mathsf{ERM}_{\mathcal{H}}(S)$ *for all $\mathcal{H}$-realizable $S \in \mathcal{Z}^*$.*
- $\mathsf{Merge} : \{0,1\}^C \times \{0,1\}^C \to \{0,1\}^C$ *such that for all $S_1, S_2 \in \mathcal{Z}^*$ such that $S_1 \cup S_2$[4] is $\mathcal{H}$-realizable, it holds that* $\mathsf{Encode}(S_1 \cup S_2) = \mathsf{Merge}(\mathsf{Encode}(S_1), \mathsf{Encode}(S_2))$.

Before we describe TiLU schemes for such classes, we list a few well-studied concept classes that are efficiently mergeable (details deferred to Appendix A).

- **Thresholds.** Class $\mathcal{H}_{\text{th}}$ consists of all *threshold functions* over $\mathcal{X} = \{1, \ldots, |\mathcal{X}|\}$, namely for any $a \in \{0, 1, \ldots, |\mathcal{X}|\}$, we have $h_{>a}(x) = \mathbb{1}\{x > a\}$; $\mathcal{H}_{\text{th}}$ is $O(\log |\mathcal{X}|)$-bit mergeable.
- **Product of d thresholds.** The class $\mathcal{H}_{\text{th}}^d$ over $\mathcal{X} = [m]^d$, consists of functions indexed by $\boldsymbol{a} = (a_1, \ldots, a_d) \in \{0, 1, \ldots, m\}^d$ defined as $h_{>\boldsymbol{a}}(x) := \mathbb{1}\left[x_1 > a_1 \wedge \cdots \wedge x_d > a_d\right]$; $\mathcal{H}_{\text{th}}^d$ is $O(d \log |\mathcal{X}|)$-bit mergeable.
- **Parities.** The class $\mathcal{H}_{\oplus}^d$ consists of all *parity functions*, namely for $\mathcal{X} = \mathbb{F}_2^d$ and $w \in \mathbb{F}_2^d$, we have $h_w(x) = \langle w, x\rangle_{\mathbb{F}_2}$; $\mathcal{H}_{\oplus}^d$ is $O(d^2)$-bit mergeable; note that $d = \log |\mathcal{X}|$.
- **Intersection-Closed Classes.** A class $\mathcal{H}$ is said to be *intersection-closed* if for all $h, h' \in \mathcal{H}$, the function $\tilde{h}$, given by $\tilde{h}(x) = h(x) \wedge h'(x)$, is also in $\mathcal{H}$. Such a class $\mathcal{H}$ is $d \log |\mathcal{X}|$-bit mergeable, where d is the VC-dimension of $\mathcal{H}$. In particular, this includes $\mathcal{H}_{\text{th}}^d$ (more examples in Appendix A).

We also consider an example of a simple class that is *not* efficiently mergeable.

- **Point Functions.** Class $\mathcal{H}_{\text{pt}}$ consists of all *point functions* over $\mathcal{X} = \{1, \ldots, |\mathcal{X}|\}$, namely for any $a \in \{1, \ldots, |\mathcal{X}|\}$, we have $h_a(x) = \mathbb{1}\{x = a\}$; $\mathcal{H}_{\text{pt}}$ is not $o(|\mathcal{X}|)$-bit mergeable.

3.1. Ticketed LU Schemes for Mergeable Concept Classes

In this section, we provide a TiLU scheme for mergeable concept classes. The basic idea is to use a *Merkle tree*-like data structure to construct tickets for each example.

Theorem 5 *For any C-bit mergeable concept class $\mathcal{H}$, there exists a TiLU scheme with space complexity* $(C_s = \log |\mathcal{H}|, C_t = O(C \log n))$.

Proof For simplicity, we assume that n is a power of 2; the argument generalizes to all n easily. Consider a full binary tree of depth $d = \log_2 n$ with leaf i corresponding to example (x_i, y_i). For each internal node v, let $S_v \triangleq \{(x_i, y_i) \mid v \text{ is an ancestor of leaf } i\}$ be the dataset consisting of

4. We use $S_1 \cup S_2$ to denote the *concatenation* of the two datasets.

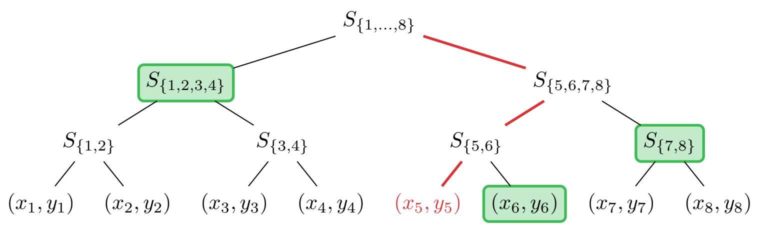


Figure 2: Illustration of TiLU scheme underlying the proof of Theorem 5. The ticket t_5 for the example (x_5, y_5) are the index i and the outputs of Encode applied on $S_{\{1,2,3,4\}}$, $S_{\{7,8\}}$, and $S_{\{6\}}$.

examples corresponding to the leaf nodes in the subtree under v. For any leaf i, let $v_1, \ldots, v_{d-1}$ be the nodes on the path from the root to leaf i, and for each $j \in \{2, \ldots, d\}$ let $\tilde{v}_j$ be the child of v_{j-1} and sibling of v_j. Figure 2 shows an example. Let the ticket corresponding to example i be given as

$$t_i \triangleq (i, \mathsf{Encode}(S_{\tilde{v}_2}), \ldots, \mathsf{Encode}(S_{\tilde{v}_d})).$$

It is immediate to see that the number of bits in t_i is $d + C \cdot (d-1)$. Define $\mathsf{Learn}(S)$ to return $h = \mathsf{Decode}(\mathsf{Encode}(S))$, $\mathsf{aux} = h$, and tickets t_i as specified above.

We define Unlearn as follows. If I is empty, then simply return h. Given a non-empty $I \subseteq [n]$, let R be the set of all nodes v such that no leaf in the sub-tree under v belongs to I, but the same is not true of the parent of v. It is easy to see that $S \smallsetminus S_I$ is precisely given as $\bigcup_{v \in R} S_v$, and moreover, S_v and $S_{v'}$ are disjoint for distinct $v, v' \in R$. For all $v \in R$, we can recover $\mathsf{Encode}(S_v)$ from ticket t_i for any leaf $i \in I$ in the subtree under the sibling of v. Thus, by repeated applications of Merge, we can recover $\mathsf{Encode}(\bigcup_{v \in R} S_v) = \mathsf{Encode}(S \smallsetminus S_I)$. Finally, we can return $h' = \mathsf{Decode}(\mathsf{Encode}(S \smallsetminus S_I))$ as the predictor after unlearning, which is identical to the predictor returned by $\mathsf{Learn}(S \smallsetminus S_I)$. ■

3.2. Improvements with Compressibility

We augment the notion of mergeable concept classes, with an additional notion of existence of compressions (similar to the notion introduced by Littlestone and Warmuth (1986)), and provide improved TiLU schemes for such classes. The advantage of this scheme over that in Theorem 5 is that the space complexity of the ticket depends on n through only an *additive* $\log n$ factor.

Definition 6 (Mergeable Concept Class with Compressions) *A concept class $\mathcal{H} \subseteq (\mathcal{X} \to \mathcal{Y})$ is said to be* C-bit mergeable with K-compressions *if in addition to* Encode, Decode, Merge *as in Definition 4, there exists a relation* $\mathsf{Compress} \subseteq \mathcal{Z}^* \times \mathcal{Z}^{\le K}$, *where we say that $T \in \mathcal{Z}^{\le K}$ is a* compression *of S if $(S, T) \in \mathsf{Compress}$. For any $\mathcal{H}$-realizable $S \in \mathcal{Z}^*$, the following properties must hold for any valid compression T: (i) $T \subseteq S$, (ii) $\mathsf{Encode}(S) = \mathsf{Encode}(T)$, and (iii) for all $z \in S \smallsetminus T$, T is a compression for $S \smallsetminus \{z\}$.*

The above notion is related, but incomparable to classes with *stable* compression schemes (Bousquet et al., 2020). A stable compression scheme requires that there exist a unique or canonical compression set for any realizable dataset, whereas in the above we allow the existence of many compressions and do not require that there be a canonical one. On the other hand, the above definition requires the mergeable property, which is not required for stable compression schemes.

It is easy to see that all the examples of mergeable concept classes we considered earlier are in fact mergeable with compressions (the details are deferred to Appendix A). In particular,

- Thresholds $\mathcal{H}_{\mathrm{th}}$ are $O(\log|\mathcal{X}|)$-mergeable with 2-compressions.
- Parities $\mathcal{H}_{\oplus}^{d}$ are $O(d^2)$-mergeable with d-compressions.
- Intersection-closed classes with VC-dimension d are $O(d\log|\mathcal{X}|)$-bit mergeable with d-compressions.

The following result concerns with TiLU schemes for mergeable concept classes with compressions.

Theorem 7 *For any class $\mathcal{H}$ that is mergeable with K-compressions, there exists a TiLU scheme with space complexity $(C_s = \log|\mathcal{H}|, C_t = 2K\log|\mathcal{Z}| + \log n)$.*

Proof Algorithm Learn works as follows. Given any dataset S, partition S into $T_1, T_2, \ldots$ such that T_i is a compression of $\bigcup_{j\geq i} T_i$ (this can be iteratively done by choosing T_1 as any compression of S, T_2 as any compression of $S \smallsetminus T_1$, and so on). The ticket t_i for the example (x_i, y_i) that lies in T_j is given as $(j, T_j \circ T_{j+1})$. The size of the ticket is $2K\log|\mathcal{Z}| + \log n$, since $|T_\ell| \leq K$ for all ℓ, and the number of parts is at most n. The predictor returned is $h = \mathsf{Decode}(\mathsf{Encode}(S))$ and $\mathsf{aux} = h$.

Algorithm $\mathsf{Unlearn}$ is defined as follows. If I is empty, then simply return h. Otherwise, let ℓ be the smallest index such that $S_I \cap T_\ell = \emptyset$. We can reconstruct T_j for all $j \leq \ell$ from the tickets, since for each $j < \ell$ there exists some example $(x_i, y_i) \in T_j$ that has been presented for unlearning, and the ticket t_i contains $T_j \circ T_{j+1}$. $\mathsf{Unlearn}$ simply returns $\mathsf{Decode}(\mathsf{Encode}(T_1 \cup \cdots \cup T_\ell \smallsetminus S_I))$. Learn on input $S \smallsetminus S_I$ would have returned $\mathsf{Decode}(\mathsf{Encode}(S \smallsetminus S_I))$. Thus, to be a valid TiLU scheme, it suffices to show that $\mathsf{Encode}(T_1 \cup \cdots \cup T_\ell \smallsetminus S_I) = \mathsf{Encode}(S \smallsetminus S_I)$. This holds because

$$\begin{aligned}\mathsf{Encode}(T_1 \cup \cdots \cup T_\ell \smallsetminus S_I) &\overset{(*)}{=} \mathsf{Merge}(\mathsf{Encode}(T_1 \cup \cdots \cup T_{\ell-1} \smallsetminus S_I), \mathsf{Encode}(T_\ell)) \\ &\overset{(**)}{=} \mathsf{Merge}(\mathsf{Encode}(T_1 \cup \cdots \cup T_{\ell-1} \smallsetminus S_I), \mathsf{Encode}(\textstyle\bigcup_{j\geq\ell} T_j \smallsetminus S_I)) = \mathsf{Encode}(S \smallsetminus S_I),\end{aligned}$$

where, $(*)$ follows from the property of Merge and $(**)$ uses that $\mathsf{Encode}(T_\ell) = \mathsf{Encode}(\bigcup_{j\geq\ell} T_j \smallsetminus S_I)$, since T_ℓ is a compression of $\bigcup_{j\geq\ell} T_j$ and hence a compression of $\bigcup_{j\geq\ell} T_j \smallsetminus S_I$. ■

4. Sharper Bounds for Specific Classes

While the previous general reductions are applicable to a large number of concept classes, the resulting space complexity bounds can be improved for certain classes by designing more specific algorithms. In this section, we present TiLU schemes for point functions and (product of d) thresholds, as stated formally below. In the case of product of d thresholds, this significantly improves the dependency on n from $\log n$ to $\log\alpha^{-1}(n)$ (inverse-Ackermann function; see Definition 9), and in case of 1D thresholds, we also improve the dependency on domain size.

Theorem 8 *There exist TiLU schemes for*

(a) $\mathcal{H}_{\mathrm{pt}}$ with space complexity $(O(\log|\mathcal{X}|), O(\log\alpha^{-1}(n)))$.

(b) $\mathcal{H}_{\mathrm{th}}^{d}$ with space complexity $(O(\log|\mathcal{X}|), O(\log|\mathcal{X}| + d\cdot\log\alpha^{-1}(n)))$.

Class $\mathcal{H}$	C_t (from Theorem 5)	C_t (from Theorem 7)	C_t (from Theorem 8)
1D Thresholds	$O(\log\lvert\mathcal{X}\rvert \cdot \log n)$	$O(\log\lvert\mathcal{X}\rvert + \log n)$	$O(\log \alpha^{-1}(n))$
Product of d thresholds	$O(d \log\lvert\mathcal{X}\rvert \cdot \log n)$	$O(d \log\lvert\mathcal{X}\rvert + \log n)$	$O(\log\lvert\mathcal{X}\rvert + d \log \alpha^{-1}(n))$
Parities ($\mathcal{X} = \mathbb{F}_2^d$)	$O(d^2 \log n)$	$O(d^2 + \log n)$	—
Point Functions	—	—	$O(\log \alpha^{-1}(n))$

Table 1: The size C_t of tickets, derived as corollaries of Theorem 5 for various concept classes. The size C_s of aux is $O(\log\lvert\mathcal{H}\rvert)$ in each case.

(c) $\mathcal{H}_{\text{th}}$ *with space complexity* $(O(\log\lvert\mathcal{X}\rvert), O(\log \alpha^{-1}(n)))$.

Note that Item (c) is an improvement over Item (b) for $d = 1$, as the ticket size in the former does not depend on $\lvert\mathcal{X}\rvert$ at all. Moreover, Item (a) provides a TiLU scheme for the class of point functions for which the techniques from Theorems 5 and 7 were not applicable (see Proposition 19). In Table 1, we summarize the implied bounds of Theorems 5, 7 and 8 for some concrete classes.

At the heart of our improvements is a problem called *count-to-zero*, which we introduce and give a TiLU scheme for in Section 4.1. In the subsequent sections, we then use it to give TiLU schemes for each of the aforementioned class. We only describe high-level overviews and the full proof of Theorem 8 is deferred to Appendix D.

4.1. Count-to-Zero

We first describe the *Count-to-Zero* (CtZ) problem. Here there are m examples, where m is unknown a priori. After receiving an unlearning request, we would like to tell whether there is still any example left (that is not unlearned). For CtZ, we follow the terminologies of Definition 2 except that Unlearn returns either $\perp$ (when $S_I = S$, and thus no example is left) or $\top$ (when $S_I \subsetneq S$, and some example is remaining), instead of the hypotheses h, h'.[5]

We give a TiLU scheme for CtZ with ticket size $O(\log \alpha^{-1}(m))$ bits (and one-bit auxiliary information), where α^{-1} is the inverse-Ackermann function defined below. Note that a naive algorithm—writing down m in each ticket—requires ticket size $O(\log m)$ bits.

Definition 9 *Consider a sequence of functions $A_1, A_2, \ldots$ defined as, $A_r(1) = 2$ for all $r \geq 1$, and for all $t \geq 2$,*

- *$A_1(t) = 2t$ (interpretable as t-fold repeated addition of 2), and*
- *$A_{r+1}(t) = A_r(A_{r+1}(t-1))$, i.e., $A_{r+1}(t) = A_r^{t-1}(2) = \underbrace{A_r(A_r(\cdots(A_r(2))))}_{t-1 \text{ times}}$.*

The Ackermann function *$\alpha(t)$ is defined as $A_t(t)$. The* inverse-Ackermann function *$\alpha^{-1}(n)$ is defined as the smallest t such that $n \leq \alpha(t)$.*

For example, $A_2(t) = 2^t$, and $A_3(t) = 2 \uparrow\uparrow t$, where $a \uparrow\uparrow k$ denotes the kth tetration of a (i.e., $a^{a^{\cdot^{\cdot^{\cdot^{a}}}}}$ of order k), etc. The Ackermann function was demonstrated as a function that is recursive, but not primitive recursive.

5. We note that the outputs $\perp$ or $\top$ can be stored using only one bit.

Theorem 10 *There is a TiLU scheme for* Count-to-Zero *(CtZ) problem with space complexity* $(C_s = 1, C_t = O(\log \alpha^{-1}(m)))$.

4.1.1. SMALL-ALPHABET SIZE-INDEXING SPERNER FAMILIES

Recall that a *Sperner family* is a family $\mathcal{Q}$ of *multi*-sets $(Q_i)_{i\in\mathbb{N}}$ such that none of them is a sub-multiset of another (see the book by Engel (1997) for background on Sperner families). We say that a Sperner family $\mathcal{Q}$ is *size-indexing* if for all $m \in \mathbb{N}$, the multiset Q_m has m elements (denoted $|Q_m| = m$). Furthermore, we say that $\mathcal{Q}$ is of *alphabet size* $n(m)$ if for all $m \in \mathbb{N}$, each element of the multiset Q_m is from the ordered alphabet $[n(m)] = \{1, \dots, n(m)\}$. For $\ell, r \in \mathbb{N}$, $\ell < r$, we define $[\ell, r]$-*size-indexing* Sperner family similar to above, except with $\mathcal{Q} = \{Q_\ell, \dots, Q_r\}$.

Lemma 11 *For all* $r, t \geq 1$, *there is an* $[A_r(t), A_r(A_r(t))]$-*size-indexing Sperner family* $\mathcal{Q}^{r,t}$ *with alphabet size* $2r$.

Proof We prove this by induction over r. Consider the base case of $r = 1$, i.e., we want to construct a $[2t, 4t]$-size-indexing Sperner family $\mathcal{Q}^{1,t}$ with alphabet $\{\texttt{1}, \texttt{2}\}$. For each $m \in \{2t, \dots, 4t\}$, let $Q^{1,t}_m$ contain $4t - m$ copies of $\texttt{1}$ and $2m - 4t$ copies of $\texttt{2}$. It is easy to see that $|Q^{1,t}_m| = m$. Furthermore, $\{Q^{1,t}_{2t}, \dots, Q^{1,t}_{4t}\}$ is a Sperner family since none of the multisets is a sub(multi)set of another multiset (i.e., $m \neq m' \implies Q^{1,t}_m \not\subseteq Q^{1,t}_{m'}$).

Next, consider the case of $r > 1$. Let the alphabet be $\{\texttt{1}, \dots, \texttt{2r}\}$. If $t = 1$, then note that $A_r(t) = 2$ and $A_r(A_r(t)) = 4$, and thus from above, there exists $\mathcal{Q}^{r,t}$ with alphabet size 2. Next, consider the case of $t \geq 2$. For ease of notation, let $T = A_r(t)$. From the inductive hypothesis we have that there exists an $[A_{r-1}(t'), A_{r-1}(A_{r-1}(t'))]$-size-indexing family $Q^{r-1,t'}$ for all $t' \geq 1$. Fix $m \in [T, A_r(T)]$ and let j_m be the unique value such that $A_r(j_m) \leq m - T + 2 < A_r(j_m + 1)$. Since $m \leq A_r(T)$ we have that $1 \leq j_m < T$. We construct $Q^{r,t}_m$ as the union of $T - j_m - 1$ copies of $\texttt{2r-1}$ and $j_m - 1$ copies of $\texttt{2r}$ and Q^{r-1,j_m}_{m-T+2}, where the latter uses symbols $\{\texttt{1}, \dots, \texttt{2r-2}\}$.

Clearly, $|Q^{r,t}_m| = m$. To see that this is a Sperner family, consider any $m < m'$. Note that since $|Q_m| = m < m' = |Q_{m'}|$ we immediately have $Q_{m'} \not\subseteq Q_m$; thus, it suffices to show that $Q_m \not\subseteq Q_{m'}$. Consider two cases:

- $\boldsymbol{j_m = j_{m'} = j}$: Since $Q_m \cap [2r-2] = Q^{r-1,j}_{m-T+1}$ and $Q_{m'} \cap [2r-2] = Q^{r-1,j}_{m'-T+2}$ are two (different) subsets from the same Sperner family $\mathcal{Q}^{r-1,j}$, and hence $Q_m \not\subseteq Q_{m'}$.
- $\boldsymbol{j_m < j_{m'}}$: Q_m contains $T - j_m - 1$ copies of $\texttt{2r-1}$, whereas $Q_{m'}$ contains $T - j_{m'} - 1$ copies of $\texttt{2r}$. Since $T - j_m - 1 > T - j_{m'} - 1$, we have $Q_m \not\subseteq Q_{m'}$.

■

Theorem 12 *There is a size-indexing Sperner family with alphabet size* $O(\alpha^{-1}(m)^3)$.

Proof We first show that there exists a $[2, A_t(t)]$-size-indexing Sperner family with alphabet size t^2. From Lemma 11, we have $[A_t(i), A_t(i+1)]$-size-indexing Sperner families $\mathcal{Q}^{t,i}$ using alphabet of size $2(t-1)$ (since $A_t(i) = A_{t-1}(A_t(i-1))$ and $A_t(i+1) = A_{t-1}(A_{t-1}(A_t(i-1)))$). Renaming the symbols to be distinct and setting $\mathcal{Q} = (\mathcal{Q}^{t,1}, \mathcal{Q}^{t,2}, \dots, \mathcal{Q}^{t,t})$, we get a $[2, A_t(t)]$-size-indexing family. In particular, this implies a $[A_{t-1}(t-1), A_t(t) - 1]$-size indexing Sperner family $\mathcal{Q}^t$.

Finally, this implies a size-indexing Sperner family (for all sizes) with alphabet size $n(m) \leq O(m^3)$ as $\mathcal{Q} = (\mathcal{Q}^t)_{t\geq 0}$, by renaming symbols so that each $\mathcal{Q}^t$ uses distinct symbols. (We use $\mathcal{Q}^0$ to denote the trivial $[1, 1]$-size-indexing family with alphabet size of 1.) ■

4.1.2. FROM SIZE-INDEXING SPERNER FAMILIES TO A TILU SCHEME

Next, we show that a size-indexing Sperner family can be used to construct a TiLU scheme for CtZ. The basic idea is to use elements of $Q_{|S|}$ as the tickets. This is formalized below.

Proof of Theorem 10 For each $m \in \mathbb{N}$, let Q_m be the multiset from the size-indexing Sperner family from Theorem 12; recall $|Q_m| = m$. Finally, for each $m \in \mathbb{N}$ and $i \in [m]$, we let q_i^m denote the i-th element in Q_m (where the elements of Q_m are ordered arbitrarily). Recall also that S is our original training set. Our algorithm is as follows:

$$\mathsf{Learn}(S) = \begin{cases} (\perp, \perp, \emptyset) & \text{if } S = \emptyset \\ (\top, \top, (q_i^{|S|})_{i \in [|S|]}) & \text{if } S \neq \emptyset, \end{cases}$$

where the auxiliary information ($\top$ or $\perp$) can be written in one bit of aux. For unlearning, we define

$$\mathsf{Unlearn}(S_I, \mathsf{aux}, (t_i)_{i \in I}) = \begin{cases} \mathsf{aux} & \text{if } S_I = \emptyset \\ \perp & \text{if } S_I \neq \emptyset \text{ and } \{t_i\}_{i \in I} = Q_{|S_I|} \\ \top & \text{if } S_I \neq \emptyset \text{ and } \{t_i\}_{i \in I} \neq Q_{|S_I|}, \end{cases}$$

where the expression $\{t_i\}_{i \in I}$ is interpreted as a multiset.

The correctness is obvious in the cases $S_I = \emptyset$ and $S_I = S$. We next show correctness when $\emptyset \subsetneq S_I \subsetneq S$. Since $\mathcal{Q}$ is a Sperner family, we have $Q_{|S_I|} \nsubseteq Q_{|S|}$. However, $\{t_i\}_{i \in I} \subseteq Q_{|S_I|}$ and thus Unlearn will return $\top$—which is the correct answer—in this case.

Finally, the space complexity claim follows from the fact that $q_i^{|S|} \subseteq [O(\alpha^{-1}(|S|)^3)]$ since the Sperner family has alphabet size $O(\alpha^{-1}(m)^3)$. Each ticket only stores a single character from an alphabet of size $O(\alpha^{-1}(m)^3)$ and hence the ticket size is $O(\log \alpha^{-1}(m))$. ■

4.2. Point Functions

We can use CtZ scheme to get a TiLU scheme for $\mathcal{H}_{\text{pt}}$, the class of point function. First, we need to define the predictor h output by $\mathsf{Learn}(S)$. If $(a, 1)$ appears in S for some $a \in \mathcal{X}$, then we must output h_a. Otherwise, we can output h_b for any $b \in \mathcal{X}$ such that $(b, 0)$ is not in S; for tie-breaking, we choose the smallest such b. The first case is easy to check: we can just use CtZ to determine whether there is still any $(a, 1)$ left after unlearning. This only requires $O(1)$-size auxiliary information and $O(\log \alpha^{-1}(n))$-size tickets as desired. Now, suppose we are in the second case, and that we start off with b^* being the smallest such that $(b^*, 0)$ is not in S. The idea here is to use the CtZ scheme to check whether, after unlearning, any $(b, 0)$ remains in the set for each $b < b^*$. Note that each example (x, y) is only a part of one such subscheme (for $b = x$) and therefore the ticket size remains $O(\log \alpha^{-1}(n))$. However, there seems to be an issue: since we run this subscheme for all $b < b^*$, we may require $O(b^*) \leq O(|\mathcal{X}|)$ bits to keep the auxiliary information. Fortunately, this turns out to be unnecessary: Observe that our CtZ scheme never uses the auxiliary information except for the case where the surviving set is empty. On the other hand, if the unlearning request does not contain $(b, 0)$ for $b < b^*$, we know that S must contain $(b, 0)$ for each $b < b^*$ (due to minimality b^*). Thus, we do not require storing any additional auxiliary information. This gives the final $C_s = O(\log |\mathcal{X}|)$ and $C_t = O(\log \alpha^{-1}(n))$ bound. A formal proof is given in Appendix D.

4.3. Product of d Thresholds

In the case of product of d thresholds $\mathcal{H}_{\text{th}}^d$, we start by deriving a primitive for computing the minimum value of the dataset. In the *MINimum VALue* (MINVAL) problem, we are given a set $S \subseteq \mathcal{X}$ where $\mathcal{X} \subseteq \mathbb{R}$ is the domain and the goal is to output $\min(S)$ if $S \neq \emptyset$ and $\perp$ if $S = \emptyset$.

We first give a TiLU scheme for MINVAL with space complexity $(O(\log|\mathcal{X}|), O(\log|\mathcal{X}| + \log\alpha^{-1}(n)))$. To understand the high-level idea, let us consider the case where the input contains only distinct values from $\mathcal{X}$. In this case, the algorithm is simple: let $\mathsf{aux} = \min(S)$ and the ticket to each x be its successor (i.e., the smallest element in S greater than x). When the minimum is unlearned, we update the minimum to its successor. The actual scheme for MINVAL is more subtle, since we need to handle repeated elements. Nonetheless, the general idea is that we additionally use the CtZ scheme to help determine whether all copies of the minimum have been unlearned.

To derive a TiLU scheme for $\mathcal{H}_{\text{th}}^d$, we use the following learning algorithm for $\mathcal{H}_{\text{th}}^d$: Output $h_{>\boldsymbol{a}}$ where, for $j \in [d]$, a_j is the minimum value among all jth coordinate of the 1-labeled samples, minus one. To compute this minimum value, we simply use the TiLU scheme for MINVAL.

4.4. Further Improvements for 1D Thresholds

Our improvement for $\mathcal{H}_{\text{th}}$ is based on an insight from the following (central) LU scheme. First, use binary search to find the threshold. Namely, we start with $a_0 = \lfloor|\mathcal{X}|/2\rfloor$. If $h_{>a_0}$ agrees with all the examples, then output h_{a_0}. Otherwise, we determine whether a_0 is too large or too small, and then adjust the search range and recurse. Suppose that on the original dataset the search path is $a_0, \dots, a_i$. Observe that, after unlearning, the output predictor must be one of $h_{>a_0}, \dots, h_{>a_i}$. Thus, we record (in aux) the empirical loss of each $h_{>a_j}$. We can then update this after seeing the unlearning set. This allows us to determine where in the binary search we stop, and thus the predictor to output. Since binary search examines $O(\log|\mathcal{X}|)$ hypotheses and recording each empirical loss uses $O(\log n)$ bits, the space complexity of this LU scheme, without tickets, is $O((\log|\mathcal{X}|)\cdot(\log n))$.

Now, to reduce the dependency on n, we can apply our CtZ scheme. The most natural attempt would be to apply the CtZ scheme on the sets of examples that each $h_{>a_j}$ errs on. However, since each example can be wrong on all of $h_{>a_0}, \dots, h_{>a_{i-1}}$, an example can receive as many as $i - 1 = O(\log|\mathcal{X}|)$ different tickets from the CtZ scheme. This results in $C_t = O((\log|\mathcal{X}|)\cdot(\log\alpha^{-1}(n)))$.

To reduce the ticket size further, observe that whether the empirical error of $h_{>a_j}$ is zero is actually just whether there is an example left between a_i and a_j. With this in mind, we partition the domain $\mathcal{X}$ at points $a_1, \dots, a_j$ and then use the CtZ scheme for each partition. Since we can determine whether each partition is empty, we can determine whether each $h_{>a_j}$ has zero empirical error after unlearning. Furthermore, since each training example belongs to just a single partition, it receives a single ticket of size $O(\log\alpha^{-1}(n))$. This concludes our high-level overview.

5. Lower Bounds

Given the mild dependence on n in the above schemes' space complexity ($O(\log\alpha^{-1}(n))$ bits), it is natural to ask whether this can be removed altogether. The main result of this section is that it cannot be removed. (In other words, at least one of C_s or C_t must grow with n.) Recall that a concept class is *non-trivial* if it contains at least two concepts.

Theorem 13 *For any non-trivial $\mathcal{H}$, there does not exist any TiLU scheme for $\mathcal{H}$ with space complexity $(C_s = O(1), C_t = O(1))$.*

Once again, our key ingredient is a lower bound on size-indexing Sperner family, which we prove in Section 5.1. We then turn this into a space complexity lower bound for CtZ (Section 5.2), before finally reducing from this to space complexity of learning any concept class (Section 5.3).

5.1. Lower Bounds for Sperner Family

We first derive lower bounds for Sperner families that are more general than size-indexing, as defined below.

Let $\boldsymbol{a} = (a_i)_{i \in \mathbb{N}}$ be any infinite (not necessarily strictly) increasing sequence of integers. We say that a Sperner family $\mathcal{Q} = (Q_i)_{i \in \mathbb{N}}$ is *$\boldsymbol{a}$-size-indexing* if $|Q_i| = a_i$ for all $i \in \mathbb{N}$. (Note that, for the sequence $a_i = i$, this coincides with the size-indexing family from Section 4.1.1.) The main result of this subsection is that any $\boldsymbol{a}$-size-indexing family must have alphabet size that grows to infinity:

Theorem 14 *For any constant $\sigma \in \mathbb{N}$ and any increasing sequence $\boldsymbol{a}$, there is no $\boldsymbol{a}$-size-indexing Sperner family with alphabet size σ.*

To prove Theorem 14, we need the following well-known result , which is a simple consequence of the so-called "infinite Ramsey theorem". Recall that a subset of elements of a partially ordered set (poset) is said to be a *chain* if each element in the subset is comparable to all other elements in the subset. A subset is an *antichain* if no two pair of elements in the subset are comparable.

Lemma 15 (e.g., Tachtsis (2016)) *Any infinite poset either contains an infinite chain or an infinite anti-chain.*

Proof of Theorem 14. For convenience, we view each multiset Q_i as a vector in $\mathbb{N}^\sigma$, and let $Q_{i,j}$ denote the number of times j appears in Q_i. We will prove the main statement by induction on σ.

Base Case. The case $\sigma = 1$ is trivial, as for any $i < j$ we have $Q_i \subseteq Q_j$. We next focus on $\sigma = 2$. Suppose for the sake of contradiction that there is an $\boldsymbol{a}$-size-indexing family $\mathcal{Q}$ with alphabet size 2. Since $|Q_1| = a_1$, we have $Q_{1,1}, Q_{1,2} \leq a_1$. Since Q_1 is not a subset of any Q_i for $i > 1$, it must be the case that either $Q_{i,1} < a_1$ or $Q_{i,2} < a_1$. Thus, by the pigeonhole principle, there must exist some indices $i_1 < i_2$ (with $i_1, i_2 \leq 2a_1$) such that either $Q_{i_1,1} = Q_{i_2,1}$ or $Q_{i_1,2} = Q_{i_2,2}$. However, this is a contradiction since it implies that $Q_{i_1} \subseteq Q_{i_2}$.

Inductive Step. Suppose that the statement holds for all $\sigma < m$ for some positive integer $m \geq 3$. Furthermore, suppose for the sake of contradiction that there exists an $\boldsymbol{a}$-size-indexing family $\mathcal{Q}$ with alphabet size m. For every $i \in \mathbb{N}$, let $b_i = (Q_{i,1}, Q_{i,2} + \cdots + Q_{i,m})$, and define the natural (partial) order $b_i \leq b_{i'}$ iff $(b_i)_1 \leq (b_{i'})_1$ and $(b_i)_2 \leq (b_{i'})_2$. Applying Lemma 15, this poset must contain an infinite chain or an infinite anti-chain. We consider these two cases separately:

- **Case I**: It contains an infinite anti-chain $(b_{i_j})_{j \in \mathbb{N}}$. Consider the family $\mathcal{S} := (S_j)_{j \in \mathbb{N}}$ where S_j has elements $1, 2$ defined by letting $S_{j,1} = Q_{i_j,1}$ and $S_{j,2} = Q_{i_j,2} + \cdots + Q_{i_j,m}$. Notice that $\mathcal{S}$ is an $(a_{i_j})_{j \in \mathbb{N}}$-size indexing family. Furthermore, since b_{i_j} is an anti-chain, this is a Sperner family. This contradicts our inductive hypothesis for $\sigma = 2$.
- **Case II**: It contains an infinite chain $(b_{i_j})_{j \in \mathbb{N}}$, i.e., where $b_{i_1} < b_{i_2} < \cdots$. Consider $\mathcal{S} := (S_j)_{j \in \mathbb{N}}$ where S_j results from throwing away all ones from Q_{i_j}. $\mathcal{S}$ is an $((b_{i_j})_2)_{j \in \mathbb{N}}$-size indexing family, and $((b_{i_j})_2)_{j \in \mathbb{N}}$ is a increasing sequence (because $(b_{i_j})_{j \in \mathbb{N}}$ forms a chain).

Finally, since $Q_{i_1,1} \leq Q_{i_2,1} \leq \cdots$, $\mathcal{S}$ must form a Sperner family[6]. Since this collection only uses alphabet $2, \dots, m$, this contradicts our inductive hypothesis for $\sigma = m - 1$.

In both cases, we arrive at a contradiction, concluding the proof. ■

5.2. From Sperner Family to CtZ

Next, we observe that the lower bound on the alphabet size for Sperner Family translates to a lower bound for CtZ. This is the "reverse reduction" of Theorem 10, but this direction requires more care as we have to handle the auxiliary information aux as well.

Lemma 16 *There is no TiLU scheme for CtZ with space complexity* $(C_s = O(1), C_t = O(1))$.

5.3. From Learning any Classes to CtZ

Finally, we observe that any TiLU scheme for the learning/unlearning problem can be converted into a TiLU scheme for CtZ with the same complexity, formalized below in Theorem 17. This, together with Lemma 16, yields Theorem 13.

Lemma 17 *For any non-trivial $\mathcal{H}$, if there exists a TiLU scheme for $\mathcal{H}$ with space complexity (C_s, C_t), then there also exists a TiLU scheme for CtZ with the same space complexity (C_s, C_t).*

6. Future Directions

In this paper, we introduce and study the ticketed model of unlearning and obtain several results. Perhaps the most basic unresolved question is to characterize the classes for which space-efficient TiLU schemes exist. For example, can we get TILU schemes for all concept classes with VC-dimension d, whose space complexity scales as $\mathrm{poly}(d, \log(n), \log(|\mathcal{X}|))$? Note that we do not know the answer to this question even for some basic classes such as linear separators.

Another tantalizing open question is to make the lower bound for CtZ (Theorem 17) quantitative. Related to this, it would also be interesting to understand whether subpolylogarithmic in n dependency is always possible for all space-efficient TiLU schemes.

We already pointed out several limitations of our model in Remark 3. Extending the TiLU model to overcome these limitations, e.g., allowing approximate ERM, seems like an interesting research direction. Another intriguing direction is to study unlearning in the agnostic setting where the dataset is not assumed to be $\mathcal{H}$-realizable. It is unclear how to obtain generic space-efficient LU schemes, such as ones in Theorem 5 and Theorem 7, in the agnostic setting; as a preliminary result, we give a TiLU scheme for agnostic 1D thresholds in Appendix C.2. Finally, it is also interesting to study the ticketed model for simpler tasks such as realizability, testing, etc; Appendix F contains initial results for agnostic 1D thresholds.

ACKNOWLEDGMENTS

Part of the work was done when AS was a student researcher at Google Research, Mountain View during summer 2022. AS thanks Surbhi Goel, Sasha Rakhlin, and Karthik Sridharan for useful discussions, and acknowledges the support of NSF through awards DMS-2031883 and DMS-1953181, as well as DOE through award DE-SC0022199.

6. Otherwise, if $S_j \subseteq S_\ell$ for some $j < \ell$, we must have $Q_{i_j} \subseteq Q_{i_\ell}$ since $Q_{i_j,1} \leq Q_{i_\ell,1}$.

References

Peter Auer and Ronald Ortner. A new PAC bound for intersection-closed concept classes. *Machine Learning*, 66(2):151–163, 2007.

Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *S & P*, 2021.

Olivier Bousquet, Steve Hanneke, Shay Moran, and Nikita Zhivotovskiy. Proper learning, Helly number, and an optimal SVM bound. In *COLT*, pages 582–609, 2020.

Jonathan Brophy and Daniel Lowd. Machine unlearning for random forests. In *ICML*, pages 1092–1104, 2021.

Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *S & P*, pages 463–480, 2015.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom B Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *USENIX Security*, 2021.

Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership inference attacks from first principles. In *S & P*, pages 1897–1914, 2022a.

Nicholas Carlini, Matthew Jagielski, Nicolas Papernot, Andreas Terzis, Florian Tramer, and Chiyuan Zhang. The privacy onion effect: Memorization is relative. In *NeurIPS*, 2022b.

Nicholas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramèr, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting training data from diffusion models. *arXiv preprint arXiv:2301.13188*, 2023a.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. In *ICLR*, 2023b.

Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. *NIPS*, 2000.

Rishav Chourasia, Neil Shah, and Reza Shokri. Forget unlearning: Towards true data-deletion in machine learning. In *ICML*, 2023.

Aloni Cohen, Adam Smith, Marika Swanberg, and Prashant Nalini Vasudevan. Control, confidentiality, and the right to be forgotten. In *TPDP*, 2022.

Jimmy Z Di, Jack Douglas, Jayadev Acharya, Gautam Kamath, and Ayush Sekhari. Hidden poison: Machine unlearning enables camouflaged poisoning attacks. In *NeurIPS ML Safety Workshop*, 2022.

Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. Lifelong anomaly detection through unlearning. In *CCS*, pages 1283–1297, 2019.

Yonatan Dukler, Benjamin Bowman, Alessandro Achille, Aditya Golatkar, Ashwin Swaminathan, and Stefano Soatto. SAFE: Machine unlearning with shard graphs. *arXiv preprint arXiv:2304.13169*, 2023.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

Thorsten Eisenhofer, Doreen Riepel, Varun Chandrasekaran, Esha Ghosh, Olga Ohrimenko, and Nicolas Papernot. Verifiable and provably secure machine unlearning. *arXiv preprint arXiv:2210.09126*, 2022.

Konrad Engel. *Sperner Theory*. Cambridge University Press, 1997.

Sanjam Garg, Shafi Goldwasser, and Prashant Nalini Vasudevan. Formalizing data deletion in the context of the right to be forgotten. In *EUROCRYPT*, pages 373–402, 2020.

Antonio Ginart, Melody Guan, Gregory Valiant, and James Y Zou. Making AI forget you: Data deletion in machine learning. In *NeurIPS*, pages 3518–3531, 2019.

Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *CVPR*, 2020.

Aditya Golatkar, Alessandro Achille, Avinash Ravichandran, Marzia Polito, and Stefano Soatto. Mixed-privacy forgetting in deep networks. In *CVPR*, pages 792–801, 2021.

Laura Graves, Vineel Nagisetty, and Vijay Ganesh. Amnesiac machine learning. In *AAAI*, pages 11516–11524, 2021.

Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. In *ICML*, pages 3832–3842, 2020.

Varun Gupta, Christopher Jung, Seth Neel, Aaron Roth, Saeed Sharifi-Malvajerdi, and Chris Waites. Adaptive machine unlearning. In *NeurIPS*, pages 16319–16330, 2021.

Jason D Hartline, Edwin S Hong, Alexander E Mohr, William R Pentney, and Emily C Rocke. Characterizing history independent data structures. *Algorithmica*, 42:57–74, 2005.

Daphne Ippolito, Florian Tramèr, Milad Nasr, Chiyuan Zhang, Matthew Jagielski, Katherine Lee, Christopher A Choquette-Choo, and Nicholas Carlini. Preventing verbatim memorization in language models gives a false sense of privacy. *arXiv preprint arXiv:2210.17546*, 2022.

Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Zou. Approximate data deletion from machine learning models: Algorithms and evaluation. In *AISTATS*, pages 2008–2016, 2021.

Masayuki Karasuyama and Ichiro Takeuchi. Multiple incremental decremental learning of support vector machines. *IEEE Transactions on Neural Networks*, 21(7):1048–1059, 2010.

Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999.

Satyapriya Krishna, Jiaqi Ma, and Himabindu Lakkaraju. Towards bridging the gaps between the right to explanation and the right to be forgotten. In *ICML*, 2023.

Nick Littlestone and Manfred Warmuth. Relating data compression and learnability. *Unpublished manuscript*, 1986.

Moni Naor and Vanessa Teague. Anti-persistence: History independent data structures. In *STOC*, pages 492–501, 2001.

Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning. In *ALT*, 2021.

Quoc Phong Nguyen, Bryan Kian Hsiang Low, and Patrick Jaillet. Variational Bayesian unlearning. In *NeurIPS*, pages 16025–16036, 2020.

Enrique Romero, Ignacio Barrio, and Lluís Belanche. Incremental and decremental learning for linear support vector machines. In *ICANN*, pages 209–218, 2007.

Ayush Sekhari, Jayadev Acharya, Gautam Kamath, and Ananda Theertha Suresh. Remember what you want to forget: Algorithms for machine unlearning. In *NeurIPS*, pages 18075–18086, 2021.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *S & P*, pages 3–18, 2017.

Vinith M Suriyakumar and Ashia C Wilson. Algorithms that approximate data removal: New results and limitations. In *NeurIPS*, 2022.

Eleftherios Tachtsis. On Ramsey's theorem and the existence of infinite chains or infinite anti-chains in infinite posets. *J. Symb. Logic*, 81(1):384–394, 2016.

Anvith Thudi, Gabriel Deza, Varun Chandrasekaran, and Nicolas Papernot. Unrolling SGD: Understanding factors influencing machine unlearning. In *EuroS&P*, pages 303–319, 2022.

Amund Tveit, Magnus Lie Hetland, and Håavard Engum. Incremental and decremental proximal support vector classification using decay coefficients. In *DaWak*, pages 422–429, 2003.

Enayat Ullah, Tung Mai, Anup Rao, Ryan A Rossi, and Raman Arora. Machine unlearning via algorithmic stability. In *COLT*, pages 4126–4142, 2021.

Santiago Zanella-Béguelin, Lukas Wutschitz, Shruti Tople, Victor Rühle, Andrew Paverd, Olga Ohrimenko, Boris Köpf, and Marc Brockschmidt. Analyzing information leakage of updates to natural language models. In *CCS*, pages 363–375, 2020.

Appendix A. Examples of Mergeable Concept Classes with Compression

We recall the function classes defined in Section 3 and show that they are mergeable concept classes with compressions (as in Definition 6).

1D Thresholds. The class $\mathcal{H}_{\mathrm{th}}$ consists of all threshold functions over $\mathcal{X} = \{1, \dots, |\mathcal{X}|\}$, namely for any $a \in \{0, 1, \dots, |\mathcal{X}|\}$, we have $h_{>a}(x) = \mathbb{1}\{x > a\}$. The class $\mathcal{H}_{\mathrm{th}}$ is $O(\log |\mathcal{X}|)$-bit mergeable with 1-compressions, as witnessed by the following methods:

- For any $\mathcal{H}_{\mathrm{th}}$-realizable dataset S, we set $\mathsf{Encode}(S) = x_-$, where x_- is the largest x_i with corresponding $y_i = 0$ or 0 if no such x_i exists.
- $\mathsf{Decode}(x_-) = h_{>x_-}$.
- $\mathsf{Merge}(x_-^1, x_-^2) = \max(x_-^1, x_-^2)$.
- $\mathsf{Compress}$: If $x_- \neq 0$ for S, then $(S, T) \in \mathsf{Compress}$ iff $T = \{(x_-, 0)\}$. Otherwise, if $x_- = 0$ for S, then $(S, T) \in \mathsf{Compress}$ iff $T = \emptyset$.

Parities. The class $\mathcal{H}_{\oplus}^d$ consists of all parity functions, namely, for $\mathcal{X} = \mathbb{F}_2^d$ and $w \in \mathbb{F}_2^d$, we have $h_w(x) = \langle w, x \rangle$. The class $\mathcal{H}_{\oplus}^d$ is $O(d^2)$-bit mergeable with d-compressions, where $d = \log |\mathcal{X}|$.

- For any $\mathcal{H}_{\oplus}^d$-realizable dataset S, we set $\mathsf{Encode}(S) = W$, where W is a representation of the set of all w such that $\langle w, x_i \rangle = y_i$ for all $(x_i, y_i) \in S$. Note that this set corresponds to an affine subspace in $\mathbb{F}_2^d$ and thus, can be represented using $O(d^2)$ bits.
- $\mathsf{Decode}(W) = h_w$ where w is the lexicographically smallest vector in W.
- $\mathsf{Merge}(W_1, W_2)$ is simply a representation of $W_1 \cap W_2$, which is also an affine subspace.
- $\mathsf{Compress}$ consists of all (S, T) such that T is a minimal subsequence of S that generates the same h_w. By standard linear algebra, it follows that there are at most d such examples.

Intersection-Closed Classes. A class $\mathcal{H}$ is said to be *intersection-closed* if for all $h, h' \in \mathcal{H}$, the "intersection" $\tilde{h}$, given by $\tilde{h}(x) = h(x) \wedge h'(x)$, is also in $\mathcal{H}$. Any intersection-closed class $\mathcal{H}$ is $d \log |\mathcal{Z}|$-bit mergeable with d-compressions, where d is the VC-dimension of $\mathcal{H}$. The key property we use to show this is Lemma 18.

Lemma 18 (Auer and Ortner (2007)) *For any intersection-closed class $\mathcal{H}$ with VC-dimension d, for all $\mathcal{H}$-realizable S, there exists a unique $h_S \in \mathcal{H}$ such that for all $h \in \mathsf{ERM}_{\mathcal{H}}(S)$, it holds for all $x \in \mathcal{X}$ that $h_S(x) = 1 \implies h(x) = 1$. Moreover, there exists $T \subseteq S$ such that $|T| \leq d$ and $y_i = 1$ for all $(x_i, y_i) \in T$ and $h_T = h_S$.*

For any intersection-closed class $\mathcal{H}$ with VC-dimension d, we can consider Encode, Decode, and Merge as follows:

- For any $\mathcal{H}$-realizable dataset S, we set $\mathsf{Encode}(S) = h_T$, where h_T is as given by Lemma 18. It is easy to see that h_T can be represented using only $d \log |\mathcal{X}|$ bits.
- $\mathsf{Decode}(\tilde{h}) = \tilde{h}$.
- $\mathsf{Merge}(\tilde{h}_1, \tilde{h}_2)$ is simply the representation of $\tilde{h}_1 \wedge \tilde{h}_2$.
- $\mathsf{Compress}$ consists of all (S, T) such that T is the minimal subsequence of S, as guaranteed to exist by Lemma 18.

We now mention some examples of intersection closed classes with VC-dimension d.

- Product of d thresholds $\mathcal{H}_{\mathrm{th}}^d$, defined in Section 3.
- Product of d thresholds in n dimensions over $\mathcal{X} = \{1, \dots, m\}^n$ given as

$$\mathcal{H}_{\mathrm{th}}^{d,n} := \left\{ h_{>\boldsymbol{a}}(x) = \mathbb{1}\left[\bigwedge_{i \in [n]} x_i > a_i\right] : \boldsymbol{a} \in \{0, \dots, m\}^n, |\{i : a_i \neq 0\}| \leq d \right\}.$$

- Intersection of half-spaces from a fixed set of possible orientations $\mathcal{V} = \{v_1, \ldots, v_d\} \in \mathbb{R}^n$:

$$\mathcal{H}_{\mathcal{V}} := \{h_B(x) = \bigcap_{v \in B} \mathbb{1}\left[\langle x, v \rangle \leq 1\right] \mid B \subseteq \mathcal{V}\}.$$

  While this involves a domain of infinite size, one could consider a finite (discretized) version of the same.
- d-point functions: $\mathcal{H}_{\text{pt}}^d := \{h_A(x) = \mathbb{1}\left[x \in A\right] \;:\; A \subseteq \mathcal{X}, |A| \leq d\}$.

A.1. Lower Bounds for Mergeable Concept Classes

Recall that the class $\mathcal{H}_{\text{pt}}$ consists of all point functions over $\mathcal{X} = \{1, \ldots, |\mathcal{X}|\}$, namely for any $a \in \mathcal{X}$, we have $h_a(x) = \mathbb{1}\{x = a\}$. In contrast to the concept classes above, we show that $\mathcal{H}_{\text{pt}}$ is *not* efficiently mergeable, as stated below. Note that this lower bound is nearly tight, since recording the entire input dataset requires only $O(|\mathcal{X}| \log n)$ bits.

Proposition 19 *$\mathcal{H}_{\text{pt}}$ is not $o(|\mathcal{X}|)$-bit mergeable.*

We prove Proposition 19 using a lower bound on one-way communication complexity of the Index_m problem.

Definition 20 *The Index_m problem is given as: Alice gets an input $X \in \{0,1\}^m$, Bob gets an input $i \in [m]$, and the goal is for Bob to output X_i with probability at least $3/4$, where Alice and Bob have access to shared randomness.*

Lemma 21 (Kremer et al. (1999)) *Any one-way (Alice $\to$ Bob) protocol for Index_m has communication $\Omega(m)$.*

Proof of Proposition 19 Let $\mathcal{X} = \{1, \ldots, m+1\}$. Suppose that $\mathcal{H}_{\text{pt}}$ is C-bit mergeable. We show that this implies a C-bit protocol for Index_m. Consider the following one-way communication protocol for Index_m using a shared random permutation $\pi : \mathcal{X} \to \mathcal{X}$.

Alice: On input $X \in \{0,1\}^m$:
- Let S_1 be a dataset containing X_j copies of $(\pi(j), 0)$ for all $j \in [m]$.
- Send $\mathsf{Encode}(S_1)$ to Bob.

Bob: On input $i \in [m]$ and message $E = \mathsf{Encode}(S_1)$ from Alice:
- Let S_2 be the dataset containing one copy of $(\pi(j), 0)$ for all $j \in [m] \setminus \{i\}$.
- If $\mathsf{Decode}(\mathsf{Merge}(E, \mathsf{Encode}(S_2))) = h_{\pi(i)}$, return $Y_i = 0$. Otherwise, return $Y_i = 1$.

Note that $S_1 \cup S_2$ is always realizable since $h_{\pi(m+1)} \in \mathsf{ERM}_{\mathcal{H}_{\text{pt}}}(S_1 \cup S_2)$. To show that this communication protocol solves Index_m, we show the following:

Claim: $\Pr[Y_i = 1 \mid X_i = 1] = 1$.
When $X_i = 1$, $(\pi(j), 0)$ appears at least once in $S_1 \cup S_2$ for all $j \in [m]$. Therefore, the unique ERM for $S_1 \cup S_2$ is $h_{\pi(m+1)}$ and not $h_{\pi(i)}$.

Claim: $\Pr[Y_i = 1 \mid X_i = 0] = 1/2$.
It suffices to prove the statement for π that is fixed on all coordinates except for $\pi(i)$ and $\pi(m+1)$. Let x_1, x_2 denote the remaining elements of $\mathcal{X}$ (to be assigned to $\pi(i), \pi(m+1)$). Once the rest of π is fixed and $X_i = 0$, $\mathsf{Decode}(\mathsf{Encode}(S_1 \cup S_2))$ is one of h_{x_1} or h_{x_2}, since $S \smallsetminus S_I$ contains $(x, 0)$ for all $x \in \mathcal{X} \smallsetminus \{x_1, x_2\}$ and so $h_x \notin \mathsf{ERM}_{\mathcal{H}_{\mathrm{pt}}}(S \smallsetminus S_I)$. Without loss of generality, suppose $\mathsf{Decode}(\mathsf{Encode}(S_1 \cup S_2)) = h_{x_1}$. Thus, we have $\Pr[Y_i = 1 \mid X_i = 1, \pi_{-\{i,m+1\}}] = \Pr[\pi(i) = x_1 \mid \pi_{-\{i,m+1\}}] = 1/2$.

We can solve Index_m with probability $3/4$ by repeating the above protocol twice, returning 1 if each step returns 1. This ensures correctness with probability $3/4$. Thus, we get that $C = \Omega(m) = \Omega(|\mathcal{X}|)$. ■

Appendix B. LU Schemes and Lower Bounds in the Central Model

In this section we discuss LU schemes and lower bounds (in the central model) for simple function classes of 1D thresholds $\mathcal{H}_{\mathrm{th}}$ and point functions $\mathcal{H}_{\mathrm{pt}}$ (see Appendix A for definitions).

B.1. LU Scheme for 1D Thresholds

Theorem 22 *There exists a LU scheme for $\mathcal{H}_{\mathrm{th}}$ with space complexity $C = O((\log |\mathcal{X}|)(\log n))$, that is valid for $\mathcal{H}_{\mathrm{th}}$-realizable datasets.*

Proof We describe the methods Learn and Unlearn, given a $\mathcal{H}_{\mathrm{th}}$-realizable dataset S and unlearning request S_I. For simplicity, we will assume that $\mathcal{X} = \{1, \ldots, 2^d - 2\}$, i.e., $|\mathcal{H}_{\mathrm{th}}| = |\mathcal{X}| + 1 = 2^d - 1$ for some integer d.

Learn. Since S is $\mathcal{H}_{\mathrm{th}}$-realizable, it follows that $\mathsf{ERM}_{\mathcal{H}_{\mathrm{th}}}(S)$ is an "interval" $\{h_{>p}, \ldots, h_{>q}\}$, where p is the largest x_i in S with $y_i = 0$ (or 0 if none exists) and $q + 1$ is the smallest x_i in S with $y_i = 1$ (or 2^d if none exists). On input S, let $h_{>a_0}, h_{>a_1}, \ldots$ be the sequence of predictors obtained when performing a binary search. Such a sequence can be obtained using the following procedure.

```
• $i \leftarrow 0$ and $a_0 \leftarrow 2^{d-1} - 1$
• while $a_i < p$ or $a_i > q$
    ▷ $a_{i+1} \leftarrow \begin{cases} a_i + 2^{d-i-2} & \text{if } a_i < p \\ a_i - 2^{d-i-2} & \text{if } a_i > q \end{cases}$
    ▷ $i \leftarrow i + 1$
• return $h_{>a_0}, h_{>a_1}, \ldots, h_{>a_i}$
```

We define $\mathsf{Learn}(S)$ to return the predictor $h_{>a_i}$ (last one in the sequence), and auxiliary information $\mathsf{aux} = (a_i, \mathrm{err}_0, \mathrm{err}_1, \ldots, \mathrm{err}_i)$, where $\mathrm{err}_j := \mathcal{L}(h_{>a_j}; S)$. Since $i \le d$, we have that the bit representation of aux is at most $O(d \cdot \log n)$.

Unlearn. For any $S_I \subseteq S$, we have that $\mathsf{ERM}_{\mathcal{H}_{\mathrm{th}}}(S \smallsetminus S_I) \supseteq \mathsf{ERM}_{\mathcal{H}_{\mathrm{th}}}(S)$. Thus, $\mathsf{ERM}_{\mathcal{H}_{\mathrm{th}}}(S \smallsetminus S_I)$ is given by the interval $\{h_{>p'}, \ldots, h_{>q'}\}$ for $p' \le p \le q \le q'$. The sequence generated by binary search on $S \smallsetminus S_I$ would be the same as $h_{>a_0}, h_{>a_1}, \ldots$ except it might stop earlier.

We define $\mathsf{Unlearn}(S_I, \mathsf{aux})$ as follows. First, using a_i, it is simple to see that we can recover $a_0, \ldots, a_{i-1}$, as there is a unique binary search path to reach a_i. Then, for $j = 0, \ldots, i$, compute

$\mathcal{L}(h_{>a_j}; S \setminus S_I) = \mathrm{err}_j - \mathcal{L}(h_{>a_j}; S_I)$, and return the predictor h_j, where j is the smallest index with $\mathcal{L}(h_{>a_j}; S \setminus S_I) = 0$. ■

We note that the unlearning algorithm in the proof of Theorem 22 even allows for multi-shot unlearning (which is not the focus of this paper).

B.2. Lower Bounds on LU Schemes for Point Functions

While we showed that the class $\mathcal{H}_{\mathrm{th}}$ admits a space-efficient LU scheme, the same is not the case for the class $\mathcal{H}_{\mathrm{pt}}$ of point functions: we show that any LU scheme must have space complexity at least linear in the number of examples. We, however, also show two ways to circumvent this barrier. First, if we augment the class $\mathcal{H}_{\mathrm{pt}}$ to $\overline{\mathcal{H}}_{\mathrm{pt}}$ to also include the *zero* function, then we can obtain a space-efficient LU scheme. Alternatively, if we only restrict to datasets without any repetitions, then again there exists a space-efficient LU scheme.

Theorem 23 *For the class $\mathcal{H}_{\mathrm{pt}}$ of point functions,*

(a) *Any LU scheme for $\mathcal{H}_{\mathrm{pt}}$, even those valid only for $\mathcal{H}_{\mathrm{pt}}$-realizable datasets, must have space complexity $\Omega(|\mathcal{X}|)$ when $n \geq \Omega(|\mathcal{X}|)$.*

(b) *There exists an LU scheme for $\overline{\mathcal{H}}_{\mathrm{pt}}$ with space complexity $C = O(\log|\mathcal{X}| + \log n)$, that is valid for $\overline{\mathcal{H}}_{\mathrm{pt}}$-realizable datasets.*

(c) *There exists an LU scheme for $\mathcal{H}_{\mathrm{pt}}$ with space complexity $C = O(\log|\mathcal{X}|)$, that is valid for $\mathcal{H}_{\mathrm{pt}}$-realizable datasets without repetitions.*

We note that part-(a) and part-(b), taken together, imply a separation between proper and improper learning in terms of the storage needed for LU schemes. While proper learning requires $\Omega(n \wedge |\mathcal{X}|)$ bits in the central memory, improper learning can be done with logarithmically many bits.

B.2.1. LOWER BOUNDS ON SPACE-COMPLEXITY OF LU SCHEMES FOR $\mathcal{H}_{\mathrm{pt}}$

We prove Theorem 23(a), using the lower bound on one-way communication complexity of the Index_m problem (Lemma 21).

Proof of Theorem 23(a). Consider an LU scheme for $\mathcal{H}_{\mathrm{pt}}$ with Learn and $\mathsf{Unlearn}$ methods. Let $\mathcal{X} = \{1, \ldots, m+1\}$. Consider the following one-way communication protocol for Index_m using a shared random permutation $\pi : \mathcal{X} \to \mathcal{X}$.

Alice: On input $X \in \{0,1\}^m$:
- Let S be a dataset containing $X_i + 1$ copies of $(\pi(i), 0)$ for all $i \in [m]$.
- Let $(h, \mathsf{aux}) \leftarrow \mathsf{Learn}(S)$
- Send aux to Bob.

Bob: On input $i \in [m]$ and message aux from Alice:
- Let $h \leftarrow \mathsf{Unlearn}(S_I, \mathsf{aux})$ for $S_I := \{(\pi(i), 0)\}$.
- If $h = h_{\pi(i)}$, return $Y_i = 0$. Otherwise, return $Y_i = 1$.

To show that this communication protocol solves Index_m, we show the following:

Claim: $\mathbf{Pr}[\boldsymbol{Y_i = 1 \mid X_i = 1}] = \mathbf{1}$.
When $X_i = 1$, $(\pi(i), 0)$ appears twice in S. Therefore, even after unlearning S_I, a copy of $(\pi(i), 0)$ remains in $S \setminus S_{I_n}$. Thus, $h_{\pi(i)}$ cannot be the ERM and therefore, $Y_i = 1$.

Claim: $\mathbf{Pr}[\boldsymbol{Y_i = 1 \mid X_i = 0}] = \mathbf{1/2}$.
It suffices to prove the statement for π that is fixed on all coordinates except for $\pi(i)$ and $\pi(m+1)$. Let x_1, x_2 denote the remaining elements of $\mathcal{X}$ (to be assigned to $\pi(i), \pi(m+1)$). Once the rest of π is fixed and $X_i = 0$, the predictor returned by $\mathsf{Learn}(S \setminus S_I)$ is one of h_{x_1} or h_{x_2}, since $S \smallsetminus S_I$ contains $(x, 0)$ for all $x \in \mathcal{X} \smallsetminus \{x_1, x_2\}$ and so $h_x \notin \mathsf{ERM}_{\mathcal{H}_{\mathrm{pt}}}(S \smallsetminus S_I)$. Without loss of generality, suppose the predictor is h_{x_1}. Thus, we have $\Pr[Y_i = 1 \mid X_i = 1, \pi_{-\{i,m+1\}}] = \Pr[\pi(i) = x_1 \mid \pi_{-\{i,m+1\}}] = 1/2$.

We can solve Index_m with probability $3/4$ by repeating the above protocol twice, returning 1 if each step returns 1. This ensures correctness with probability $3/4$. Thus, we get that the bit complexity of aux must be $\Omega(m) = \Omega(|\mathcal{X}|)$. ■

B.2.2. LU Scheme for Point Functions Augmented with Zero

Proof of Theorem 23(b). Recall that $\overline{\mathcal{H}}_{\mathrm{pt}}$ consists of functions $h_a(x) = \mathbb{1}\{x = a\}$ for all $a \in \mathcal{X}$, as well as, $h_0(x) = 0$ for all $x \in X$. The key idea is that on input S, the learning algorithm returns a predictor h_a if the example $(a, 1)$ appears in the dataset, else returns the predictor h_0.

Learn. On the input S, let $h = h_a$ if example $(a, 1)$ appears in S for some a, otherwise if all the labels are 0, let $h = h_0$. Note that there can be at most one such a since S is $\mathcal{H}_{\mathrm{pt}}$-realizable.

When $h = h_a$, the example $(a, 1)$ can appear more than once. So we set $\mathsf{aux} = (h, c)$, where c is the number of times $(a, 1)$ appears in S, or 0 if $h = h_0$. Thus, the bit representation of aux is $\log |\mathcal{H}| + \log n$.

Unlearn. On input $S_I \subseteq S$, and $\mathsf{aux} = (h, c)$ if $h = h_0$ return h_0. Else, if $h = h_a$ and the number of times $(a, 1)$ appears in S_I is equal to c, then return h_0, else return h_0. Basically, using aux, we are able to verify if $S \smallsetminus S_I$ contains any example with label 1 or not. ■

B.2.3. LU Scheme for Datasets Without Repetitions

Proof of Theorem 23(c). The key idea is that on input S, the learning algorithm returns the predictor h_a where either $(a, 1)$ appears in S or a is the smallest value such that $(a, 0)$ does not appear in S. With that in mind, we define Learn and $\mathsf{Unlearn}$ as follows.

Learn. On input S,
- If $(a, 1)$ appears in S for some a, then return the predictor h_a and $\mathsf{aux} = (a, b)$, where b is the smallest value such that $(b, 0)$ does not appear in S.
- If all labels in S are 0, then let a be the smallest value such that $(a, 0)$ does not appear in S, and return the predictor h_a and $\mathsf{aux} = (a, a)$.

It is easy to see that the bit complexity of aux is $2 \log |\mathcal{X}|$.

Unlearn. On input S_I and $\mathsf{aux} = (a, b)$, return h_a if S_I is empty. Otherwise,

- if $a \neq b$ and $(a, 1) \notin S_I$, then return h_a.
- else, let c be the smallest value such that $(c, 0) \in S_I$ return $h_{\min\{b,c\}}$.

■

Appendix C. Separation between Central and Ticketed LU Models: Agnostic Case

Note that Theorem 8 and Theorem 23 show that the class of point functions do not have space-efficient LU schemes, but do have space-efficient TiLU schemes. In this section, we provide yet another example for which there is a space-efficient TiLU scheme, but there does not exist any space-efficient LU scheme. In particular, this example holds for the class of 1D thresholds in the *agnostic* setting. We show the following:

Theorem 24 *For the class $\mathcal{H}_{\mathrm{th}}$ of 1D thresholds,*

(a) *Any LU scheme that is valid for all (even unrealizable) datasets must have space complexity $\Omega(\min\{n, |\mathcal{X}|\})$.*

(b) *There exists a TiLU scheme valid for all (even unrealizable) datasets with space complexity $(C_s = \log|\mathcal{X}|, C_t = \log|\mathcal{X}|(\log|\mathcal{X}| + \log n))$.*

In contrast, if S is $\mathcal{H}_{\mathrm{th}}$-realizable, then Theorem 22 shows a space-efficient LU scheme in the central model.

C.1. Lower Bounds for Central LU Schemes

Suppose for contradiction we have an LU scheme for $\mathcal{H}_{\mathrm{th}}$, consisting of Learn and $\mathsf{Unlearn}$ methods, that is valid even in the agnostic setting. Suppose $|\mathcal{X}| \geq 4m + 4$. We will construct a two-party one-way communication protocol, with communication complexity being the space complexity of the LU scheme, and where Bob can entirely reconstruct Alice's input $X \in \{0,1\}^m$ (with probability 1).

Alice: On input $X \in \{0,1\}^m$

- Construct a dataset S as follows:
  - For each $i \in [m]$,
    - ∗ if $X_i = 0$ include examples $(4i + 1, 1)$, $(4i + 2, 0)$, $(4i + 3, 1)$, $(4i + 4, 0)$.
    - ∗ if $X_i = 1$ include examples $(4i + 1, 1)$, $(4i + 2, 1)$, $(4i + 3, 0)$, $(4i + 4, 0)$.
- Let $(h, \mathsf{aux}) \leftarrow \mathsf{Learn}(S)$ and send aux to Bob.

Bob: On receiving aux from Alice, construct $Y \in \{0,1\}^m$ as follows:

- For each $i \in [m]$:
  - Let $h^{(i)} \leftarrow \mathsf{Unlearn}(S_{I_i}, \mathsf{aux})$ for $S_{I_i} := ((4i + 1, 1), (4i + 4, 0))$.
  - If $h^{(i)} = h_{>4i+2}$, set $Y_i = 0$. Otherwise, set $Y_i = 1$.

If $X_i = 0$, then after unlearning S_{I_i}, the only $h \in \mathsf{ERM}_{\mathcal{H}_{\mathrm{th}}}(S \smallsetminus S_{I_i})$ is $h_{>4i+2}$. On the other hand, if $X_i = 1$, then after unlearning S_{I_i}, $h_{>4i+2} \notin \mathsf{ERM}_{\mathcal{H}_{\mathrm{th}}}(S \smallsetminus S_{I_i})$. This reduction is visualized in Figure 3.

Thus, Bob can perfectly recover Alice's input X. Hence, we require that aux contains at least m bits.

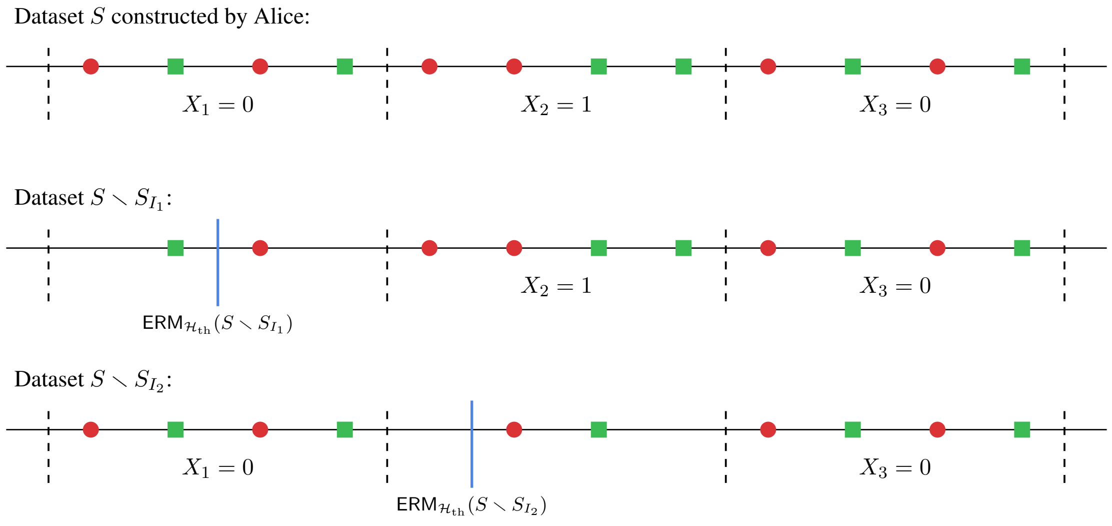

Dataset S constructed by Alice:
$X_1 = 0$
$X_2 = 1$
$X_3 = 0$
Dataset $S \setminus S_{I_1}$:
$\mathsf{ERM}_{\mathcal{H}_{\rm th}}(S \setminus S_{I_1})$
Dataset $S \setminus S_{I_2}$:
$\mathsf{ERM}_{\mathcal{H}_{\rm th}}(S \setminus S_{I_2})$

Figure 3: Illustration of lower bound reduction in Theorem 24, with red circles for label 1 and green squares for label 0.

C.2. Upper Bounds for TiLU Schemes

We now describe a TiLU scheme for $\mathcal{H}_{\rm th}$ with space complexity $(C_s = \log|\mathcal{X}|, C_t = \mathrm{poly}(\log n, \log|\mathcal{X}|))$ that is valid for all distributions. We first describe the key ideas before delving into the finer details. We will construct a Learn algorithm that would return the predictor $h_{>a} \in \mathsf{ERM}_{\mathcal{H}_{\rm th}}(S)$ with the smallest possible a (hence forth referred to as the "minimal ERM").

For simplicity we assume that $|\mathcal{X}| = D = 2^d$ is a power of 2 (the proof can easily be generalized to other values). We use the following notation below for all $p, q \in \mathcal{X} = \{1, \ldots, D\}$ with $p < q$:

- $S_{p,q}$ denotes dataset within S with x-values in $\{p, \ldots, q\}$.
- Let $a_{p,q}$ be the smallest value such that $h_{>a_{p,q}} \in \mathsf{ERM}_{\mathcal{H}_{\rm th}}(S_{p,q})$ and $p - 1 \le a_{p,q} \le q$; note that such a value exists because all examples in $S_{p,q}$ have x values between p and q due to which $\mathcal{L}(h_{>a}; S_{p,q})$ is same for all $a \le p - 1$ and similarly, the same for all $a \ge q$.
- $\mathrm{err}_{p,q} := \mathcal{L}(h_{>a_{p,q}}; S)$. Intuitively speaking, $\mathrm{err}_{p,q}$ is the smallest loss incurred by any function $h_{>a}$ over S subject to $p - 1 \le a \le q$.

Consider an interval $[p, q]$ such that no example in S_I has x-value in $[p, q]$. Now, if the minimal ERM of $S \setminus S_I$ happens to be of the form $h_{>a}$ for $p - 1 \le a \le q$, then, the minimal ERM has to be $h_{>a_{p,q}}$. Moreover, we have $\mathcal{L}(h_{>a_{p,q}}; S \setminus S_I) = \mathrm{err}_{p,q} - \mathcal{L}(h_{>a_{p,q}}; S_I)$.

Learn. Consider a full binary tree of depth d with leaf x corresponding to a possible input value $x \in \mathcal{X}$. For each internal node v, let p_v be the smallest leaf index under v, and similarly let q_v be the largest leaf index under v. For each example (x_i, y_i), let $v_1, \ldots, v_{d-1}$ be the nodes on the path from the root to leaf x_i, and for each $j \in \{2, \ldots, d\}$ let $\tilde{v}_j$ be the child of v_{j-1} and sibling of v_j. Figure 4 shows an example. Let the ticket corresponding to example (x_i, y_i) be given as

$$t_i \triangleq ((a_{p_{v_2}, q_{v_2}}, \mathrm{err}_{p_{v_2}, q_{v_2}}), \ldots, (a_{p_{v_d}, q_{v_d}}, \mathrm{err}_{p_{v_d}, q_{v_d}}))$$

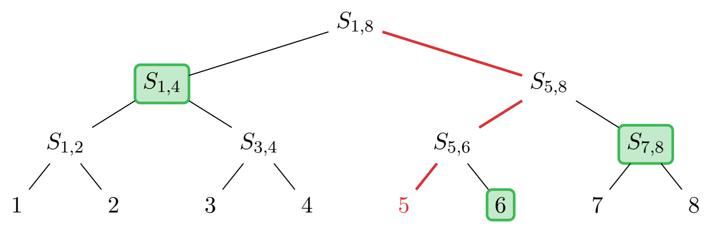


Figure 4: Illustration of TiLU scheme underlying the proof of Theorem 24(b) for $|\mathcal{X}| = 8$. The ticket t_i for all examples of the form $(x_i = 5, y_i)$ are the predictors $h_{>a_6}$, $h_{>a_{7,8}}$, $h_{>a_{1,4}}$ as well as err_6, $\mathrm{err}_{7,8}$ and $\mathrm{err}_{1,4}$.

It is immediate to see that the number of bits in t_i is $d(d + \log n)$. Define $\mathsf{Learn}(S)$ to return $h = h_{>a_{1,D}}$, $\mathsf{aux} = h$, and tickets t_i as specified above.

Unlearn. If S_I is empty, then simply return h (present in aux). Given a non-empty dataset S_I indexed by $I \subseteq [n]$, we can use the tickets to construct a partition of $[1, D]$ into disjoint intervals such that either each interval is a singleton $\{p\}$ such that some example in S_I has x-value equal to p, or is of the form $[p, q]$ such that no example in S_I has an x-value in $[p, q]$. As argued above, we know that the minimal ERM for $S \smallsetminus S_I$ must be of the form $h_{>a_{p,q}}$ for one of these parts as recovered above. Moreover, it is possible to compute the loss of each such predictor over S as $\mathcal{L}(h_{>a_{p,q}}; S \smallsetminus S_I) = \mathrm{err}_{p,q} - \mathcal{L}(h_{>a_{p,q}}; S_I)$. Thus, we can choose the minimal ERM for $S \smallsetminus S_I$ by choosing the predictor $h_{>a}$ with smallest empirical loss over $S \smallsetminus S_I$ and the smallest possible value of a among the candidates above.

Appendix D. Missing Proofs from Section 4

D.1. Point Functions

Proof of Theorem 8(a) In the following, we provide the methods Learn and $\mathsf{Unlearn}$. Our algorithm will use the TiLU scheme $(\widetilde{\mathsf{Learn}}, \widetilde{\mathsf{Unlearn}})$ for CtZ from Theorem 10. One specific property of this scheme we will use is that $\widetilde{\mathsf{Unlearn}}$ does not need aux in the case where the unlearning set is non-empty.

Learn. We will separately define the hypothesis h, the auxiliary information aux, and the ticket $(t_i)_{i\in[n]}$. Recall that S is the input set of training examples.

- The hypothesis h is defined as follows:

$$h = \begin{cases} h_a & \text{if } \exists a \in \mathcal{X}, (a, 1) \in S, \\ h_{\min\{b \mid (b,0) \notin S\}} & \text{otherwise.} \end{cases}$$

- The auxiliary information aux consists of three parts $\mathsf{aux}_1, \mathsf{aux}_2, \mathsf{aux}_3$. The first part aux_1 is simply h. The second part aux_2 is one bit and records which of the two cases we are in for h, i.e., $\mathsf{aux}_2 := \mathbb{1}\{\exists a \in \mathcal{X}, (a, 1) \in S\}$. The last part aux_3 records $\min\{b \mid (b, 0) \notin S\}$.

- As for the tickets, for every $a \in \mathcal{X}$, let $S^{(a)}$ denote $\{(x_i, y_i) \mid i \in [n], x_i = a\}$. For every $a \in \mathcal{X}$, we run $\widetilde{\mathsf{Learn}}$ on $S^{(a)}$ and let the tickets for the examples in $S^{(a)}$ be as the output of $\widetilde{\mathsf{Learn}}$.

Unlearn. Unlearn proceeds as follows, based on if $\mathsf{aux}_2 = 1$. Here, for every $a \in \mathcal{X}$, let $S_I^{(a)}$ denote $\{(x_i, y_i) \mid i \in I, x_i = a\}$.

- Case I: $\mathsf{aux}_2 = 1$. Let $\mathsf{aux}_1 = h_a$. In this case, start by checking if $S_I^{(a)} = \emptyset$.
  - If $S_I^{(a)} = \emptyset$, then output h_a.
  - If $S_I^{(a)} \neq \emptyset$, then run $\widetilde{\mathsf{Unlearn}}$ on the tickets corresponding to $S_I^{(a)}$ to determine if $S_I^{(a)} = S^{(a)}$.
    * If $S_I^{(a)} \neq S^{(a)}$ (i.e., $\widetilde{\mathsf{Unlearn}}$ outputs $\top$), then output h_a.
    * If $S_I^{(a)} = S^{(a)}$ (i.e., $\widetilde{\mathsf{Unlearn}}$ outputs $\perp$), then proceed to Case II.
- Case II: $\mathsf{aux}_2 = 0$. In this case, start with $b = \mathsf{aux}_3$. Then, for $j = b-1, \ldots, 1$, do the following:
  - If $S_I^{(j)} = \emptyset$, then skip to the next j.
  - Otherwise, if $S_I^{(j)} \neq \emptyset$, then run $\widetilde{\mathsf{Unlearn}}$ on the tickets corresponding to $S_I^{(j)}$ to determine if $S_I^{(j)} = S^{(j)}$. If $S_I^{(j)} = S^{(j)}$, then update $b \leftarrow j$.

  Finally, output h_b.

To verify that this is a valid learning scheme, note that for a and j's used in the algorithm, we know that $S^{(a)}$ and $S^{(j)}$ are not empty. Therefore, by considering the case $S_I^{(a)}, S_I^{(j)} \neq \emptyset$ and uses $\widetilde{\mathsf{Unlearn}}$, we can determine whether $S^{(a)} = S_I^{(a)}$ or $S^{(j)} = S_I^{(j)}$. Due to this, our procedure correctly updates (i) if there is any 1-labeled example remaining and (ii) the minimum b whose $(b, 0)$ does not appear in the dataset. From this, it is not hard to see that $(\mathsf{Learn}, \mathsf{Unlearn})$ is a valid ticketed LU scheme. The claimed space complexity the follows immediately from that of Theorem 10. ■

D.2. Product of d Thresholds

D.2.1. MINIMUM VALUE PRIMITIVE

Theorem 25 *There is a TiLU scheme for* MINVAL *with space complexity* $(O(\log |\mathcal{X}|), O(\log |\mathcal{X}| + \log \alpha^{-1}(n)))$.

Proof For each $a \in \mathcal{X}$, we write $\mathrm{succ}_S(a)$ to denote the maximum value in S that is larger than a; if such a value does not exist, then let $\mathrm{succ}_S(a) = \perp$. Similarly, we use the convention that $\min(\emptyset) = \perp$. Similar to before, we use $S^{(a)}$ (resp. $S_I^{(a)}$) to denote $\{x_i \mid i \in [n], x_i = a\}$ (resp. $\{x_i \mid i \in I, x_i = a\}$), and let $(\widetilde{\mathsf{Learn}}, \widetilde{\mathsf{Unlearn}})$ be the ticketed LU scheme for CtZ from Theorem 10. We now describe our algorithms.

Learn. First, we let $h = \mathsf{aux} = \min(S)$. Each ticket t_i consists of two components $t_i^{(1)}, t_i^{(2)}$. The second component $t_i^{(2)}$ is simply set to $\mathrm{succ}_S(x_i)$. As for the first component, it is computed as follows: for every $a \in \mathcal{X}$, we run $\widetilde{\mathsf{Learn}}(S^{(a)})$ and assign the tickets back to samples in $S^{(a)}$ as a first component.

Unlearn. The algorithm works as follows. Start by letting b be such that $\mathsf{aux} = h_b$. Then, do the following:

- If $b = \perp$ or $S_I^{(b)} = \emptyset$, then output b and stop.
- Otherwise, if $S_I^{(b)} \neq \emptyset$, then run $\widetilde{\mathsf{Unlearn}}$ on the second component of the tickets corresponding to $S_I^{(b)}$ to determine whether $S_I^{(b)} = S^{(b)}$. If $S_I^{(b)} \neq S^{(b)}$, then output b and stop. Otherwise, if $S_I^{(b)} = S^{(b)}$, then update $b \leftarrow \mathrm{succ}_S(b)$ where $\mathrm{succ}_S(b)$ is taken from the first component of the tickets, and repeat this step.

To see the correctness, once again note that we are simply checking in each iteration whether b still belong to $S \setminus S_I$ and otherwise we let $b \leftarrow \mathrm{succ}_S(b)$. The correctness of the check follows similar to the previous proof. Moreover, given the check is correct, it is clear that the entire algorithm is correct. ■

D.2.2. FROM MINIMUM VALUE TO PRODUCT OF d THRESHOLDS

Proof of Theorem 8(*b*) Let $(\widetilde{\mathsf{Learn}}, \widetilde{\mathsf{Unlearn}})$ denote the TiLU scheme for MINVAL from Theorem 25. The algorithms work as follows.

Learn. Let $S^{(+)}$ denote the set $\{(x_i, y_i) \mid i \in [n], y_i = 1\}$ and for each $j \in [d]$, let $S_j^{(+)}$ denote $\{(x_i)_j \mid (x_i, y_i) \in S^{(+)}\}$. The output hypothesis is h_a where

$$a_j = \begin{cases} m_j & \text{if } S^{(+)} = \emptyset, \\ \min(S_j^{(+)}) - 1 & \text{otherwise.} \end{cases}$$

As for the ticket, if $y_i = 0$, then the ticket t_i is empty. For the 1-labeled samples, their ticket t_i is the concatenation of the corresponding tickets from $\widetilde{\mathsf{Unlearn}}(S_1^{(+)}), \dots, \widetilde{\mathsf{Unlearn}}(S_d^{(+)})$. Similarly, the auxiliary information is the concatenation of the auxiliary information from these $\widetilde{\mathsf{Unlearn}}$ executions.

Unlearn. We use $\widetilde{\mathsf{Unlearn}}$ to find $b_j = \min(S_j^{(+)} \setminus (S_I)_j^{(+)})$. We then let

$$a_j = \begin{cases} m_j & \text{if } b_j = \perp, \\ b_j - 1 & \text{otherwise.} \end{cases}$$

Finally, output $h_{>\boldsymbol{a}}$.

It is simple to see that the learning algorithm is correct: by definition, all 1-labeled examples are correctly labeled by $h_{>\boldsymbol{a}}$. Furthermore, if the input training set is realizable by $h_{>\boldsymbol{a}^*}$, then we must have $a_j^* \leq a_j$ for all $j \in [d]$; this implies that $h_{>\boldsymbol{a}}$ also label all 0-labeled examples correctly. The correctness of the unlearning algorithm then immediately follows from the correctness of $(\widetilde{\mathsf{Learn}}, \widetilde{\mathsf{Unlearn}})$ for MINVAL.

Since the ticket is a concatenation of the d tickets for MINVAL, we can use the bound in Theorem 25. This implies that the total ticket size is $O\left(d \cdot \left(\log m + \log \alpha^{-1}(n)\right)\right) = O(\log |\mathcal{X}| + d \cdot \log \alpha^{-1}(n)))$ as desired. A similar calculation shows that the auxiliary information has size $O(\log |\mathcal{X}|)$. ■

D.3. Thresholds

Proof of Theorem 8(c) Let $(\widetilde{\mathsf{Learn}}, \widetilde{\mathsf{Unlearn}})$ be the TiLU scheme for CtZ from Theorem 10. For any interval $[\ell, r]$, we write $S^{[\ell,r]}$ (resp. $S_I^{[\ell,r]}$) to denote $\{(x_i, y_i) \in S \mid x_i \in [\ell, r]\}$ (resp. $\{(x_i, y_i) \in S_I \mid x_i \in [\ell, r]\}$).

Learn. Let $a_0, \dots, a_i$ be the same as in the proof of Theorem 22. Let $j_0, \dots, j_i$ be such that $a_{j_0} < \dots < a_{j_i}$ Then, for each $k = 0, \dots, i-1$, we run $\widetilde{\mathsf{Learn}}(S^{[a_{j_k}+1, a_{j_{k+1}}]})$ to get aux_k and the tickets, which we assign to the corresponding examples. The final auxiliary information is then the concatenation of $\mathsf{aux}_1, \dots, \mathsf{aux}_{i-1}$.

Unlearn. Similar to the proof of Theorem 22, the only task for us is to determine whether $\mathcal{L}(h_{>a_j}; S \smallsetminus S_I) = 0$ for each $j = 0, \dots, i-1$. To do this, observe that $\mathcal{L}(h_{>a_j}; S \smallsetminus S_I) = 0$ iff $S \smallsetminus S_I$ contains no point in $[a_j + 1, a_i] \cup [a_i + 1, a_j]$. We can check this by first running $\widetilde{\mathsf{Unlearn}}$ on each of $U^{[a_{j_k}+1, a_{j_{k+1}}]}$ with auxiliary information aux_k to check whether $S \smallsetminus S_I$ contains any point in $[a_{j_k} + 1, a_{j_{k+1}}]$. This information is sufficient to determine whether $S \smallsetminus S_I$ contains any point in $[a_j + 1, a_i] \cup [a_i + 1, a_j]$. ■

Appendix E. Missing Proofs from Section 5

Proof of Lemma 16 Suppose for the sake of contradiction that there is a TiLU scheme $(\mathsf{Learn}, \mathsf{Unlearn})$ for CtZ with space complexity $(C_s = c_s, C_t = c_t)$ where both $c_s, c_t \in \mathbb{N}$ are absolute constants.

For every $p \in \{0,1\}^{c_s}$, let A^p denote the set of $a \in \mathbb{N}$ such that, if we feed in a elements to Learn as input, the returned aux is equal to p. Since $\bigcup_{p \in \{0,1\}^{c_s}} A^p = \mathbb{N} \cup \{0\}$, there must exist p^* such that A^{p^*} is infinite. Let $a_1 < a_2 < \cdots$ be the elements of A^{p^*} sorted in increasing order.

Next, for every $i \in \mathbb{N}$, let Q_i denote the set of all a_i tickets produced by Learn when the input contains a_i elements. Since $\{Q_i\}_{i \in \mathbb{N}}$ is an $\boldsymbol{a}$-size-indexing family with alphabet size $\sigma = 2^{c_t}$, Theorem 14 implies that it cannot be a Sperner family. In other words, there must exists $k < \ell$ such that $Q_k \subseteq Q_\ell$. Now, consider the Unlearn algorithm when it receives $\mathsf{aux} = p^*$ and tickets being Q_k. If it answers $\perp$, then this is incorrect for the case where we start with a_ℓ elements and removes a_k elements with the tickets corresponding to Q_k. However, if it answers $\top$, then it is also incorrect for the case where we start with a_k elements and remove everything. This is a contradiction. ■

Proof of Theorem 17 Let $(\mathsf{Learn}, \mathsf{Unlearn})$ be any TiLU scheme for $\mathcal{H}$. Let h_1 denote the predictor output by $\mathsf{Learn}(\emptyset)$ and h_2 denote any other hypothesis in $\mathcal{H}$, and let x be such that $h_1(x) \neq h_2(x)$.

Our TiLU scheme for CtZ $(\widetilde{\mathsf{Learn}}, \widetilde{\mathsf{Unlearn}})$ works as follows. $\widetilde{\mathsf{Learn}}(S)$ returns the same auxiliary information and tickets as $\mathsf{Learn}((x, h_2(x)), \dots, (x, h_2(x)))$, where $(x, h_2(x))$ is repeated $|S|$ times. $\widetilde{\mathsf{Unlearn}}(S_I, \mathsf{aux}, (t_i)_{i \in I})$ first runs $\mathsf{Unlearn}(((x, h_2(x)), \dots, (x, h_2(x))), \mathsf{aux}, (t_i)_{i \in I})$ where $(x, h_2(x))$ is repeated $|S_I|$ times, to get a predictor h'. Then, it outputs $\top$ iff $h'(x) = h_2(x)$.

It is obvious that the space complexity of $(\widetilde{\mathsf{Learn}}, \widetilde{\mathsf{Unlearn}})$ is the same as $(\mathsf{Learn}, \mathsf{Unlearn})$. For the correctness, note that if $S_I = S$, then Unlearn outputs h_1 (by our choice of h_1) and thus $\widetilde{\mathsf{Unlearn}}$ correctly outputs $\perp$. On the other hand, if $S_I \neq S$, then at least one copy of $(x, h_2(x))$ remains and therefore we must have $h'(x) = h_2(x)$. Hence, in this case, $\widetilde{\mathsf{Unlearn}}$ outputs $\top$ as desired. ■

Appendix F. Realizability Testing

In this section, we describe the *realizability testing* problem, where the goal is to test whether a given dataset S is $\mathcal{H}$-realizable, i.e., there exists $h^\star \in \mathcal{H}$ s.t. $h^\star(x) = y$ for all $(x, y) \in S$. We extend the terminologies of Definition 2 except that Learn and $\mathsf{Unlearn}$ now return $\perp$ (when S is realizable), or $\top$ (when S is not realizable by $\mathcal{H}$), instead of the hypotheses h, h'.

The easier scenario is when S is $\mathcal{H}$-realizable, in which case, for any $S_I \subseteq S$, the remaining dataset $S \setminus S_I$ is also $\mathcal{H}$-realizable. However, when S is not $\mathcal{H}$-realizable, removing S_I may make $S \setminus S_I$ to be $\mathcal{H}$-realizable, and this is the challenging scenario for realizability testing.

Similar to the rest of the results in the paper, we show a separation between central LU and TiLU schemes for realizability testing, by considering the class of 1D thresholds.

Theorem 26 *For realizability testing for the class $\mathcal{H}_{\mathrm{th}}$ of 1D thresholds,*

(a) *Any LU scheme that is valid for all datasets must have space complexity $\Omega(\min\{n, |\mathcal{X}|\})$.*

(b) *There exists a TiLU scheme valid for all datasets with space complexity $(C_s = O(\log|\mathcal{X}|), C_t = O(\log|\mathcal{X}| + \log \alpha^{-1}(n)))$.*

Before presenting the proof of Theorem 26, we first describe a TiLU scheme for *MAXimum VALue* (MAXVAL) problem, which will be a useful primitive for developing TiLU schemes for realizability testing in 1D-thresholds. In the MAXVAL problem, we are given a set $S \subseteq \mathcal{X}$ where $\mathcal{X} \subseteq \mathbb{R}$ is the domain and the goal is to output $\max(S)$ if $S \neq \emptyset$ and $\perp$ if $S = \emptyset$.

Theorem 27 *There is a TiLU scheme for* MAXVAL *with space complexity* $(O(\log|\mathcal{X}|), O(\log|\mathcal{X}| + \log \alpha^{-1}(n)))$.

Proof The scheme follows similar to the TiLU scheme for MINVAL problem given in Theorem 25, and is skipped for conciseness. ∎

Proof of Theorem 26. (a) Suppose for contradiction, we have a LU scheme in the central model for $\mathcal{H}_{\mathrm{th}}$-realizability testing, consisting of Learn and $\mathsf{Unlearn}$ methods, that is valid for all datasets. Suppose $|\mathcal{X}| \geq n$, and let $m = n/2$. We will construct a two-party one-way communication protocol, with communication complexity being the space complexity of the LU scheme, and where Bob can entirely reconstruct Alice's input $X \in \{0,1\}^m$ (with probability 1).

Alice: On input $X \in \{0,1\}^m$,

- Construct a dataset S as follows: For each $i \in [m]$, include examples $(2i+1, 0)$ and $(2i + X_i + 1, 1)$
- Let $(h, \mathsf{aux}) \leftarrow \mathsf{Learn}(S)$ and send aux to Bob.

Bob: On receiving aux from Alice, construct $Y \in \{0,1\}^m$ as follows:

- For each $i \in [m]$:
  - Let $v^{(i)} \leftarrow \mathsf{Unlearn}(S_{I_i}, \mathsf{aux})$ for $S_{I_i} := \{(2i+1, 0)\}_{i \in [2m] \setminus \{i\}} \cup \{(2j + Y_j + 1, 1)\}_{j \in [i-1]}$.
  - If $v^{(i)} = \top$ (non-realizable), set $Y_i = 0$. Else if $v^{(i)} = \perp$ (realizable), set $Y_i = 1$.
- Return Y.

The proof proceeds by arguing that Bob can successfully recover $Y = X$, which implies that aux must contain at least m bits. To prove the correctness of the construction of Y, we can use induction on i. Suppose that we have constructed $Y_1, \ldots, Y_{i-1}$ correctly. Then, in iteration i, Bob makes an unlearning request S_{I_i} to remove all 0-labeled samples except $(2i+1, 0)$, and all 1-labeled samples with x values smaller than $2i+1$ There are two cases:

- $X_i = 0$: In this case, $S \setminus S_{I_i}$ contains $(2i+1, 1)$ and is not $\mathcal{H}_{\mathrm{th}}$-realizable, meaning that we set $Y_i = 0$.
- $X_i = 1$: In this case, $S \setminus S_{I_i}$ does not contain $(2i+1, 1)$ and thus is $\mathcal{H}_{\mathrm{th}}$-realizable, leading to $Y_i = 1$.

Therefore, $Y_i = X_i$ in both cases as desired. The lower bound follows since $n = 2m$.

(b) To obtain a TiLU scheme, the key idea is that in order to check whether S is realizable via a threshold, we only need to verify if the largest 0-labeled sample is strictly smaller than the smallest 1-labeled sample. The smallest 1-labeled sample can be computed using the MINVAL primitive in Theorem 25, and the largest 0-labeled sample can be computed using the MAXVAL primitive in Theorem 27, both of which have a TiLU scheme with space complexity $(O(\log|\mathcal{X}|), O(\log|\mathcal{X}| + \log\alpha^{-1}(n)))$.

With that in mind, we define Learn and Unlearn as follows.

Learn. On input S,

- Compute S^0 as the set of all 0-labeled samples in S, and $S^1 = S \setminus S^0$.
- Implement MAXVAL on S^0, let $x^0 = \max(S^0)$ and compute the corresponding tickets for samples in S^0.
- Implement MINVAL on S^1, let $x^1 = \min(S^1)$ and compute the corresponding tickets for samples in S^1.
- If either x^0 or x^1 is $\perp$, or if $x^0 < x^1$, set $\mathsf{aux} = (\perp, x^0, x^1)$, and return.
- Else, set $\mathsf{aux} = (\top, x^0, x^1)$ and return.

Learn. On input $\mathsf{aux} = (a, x^0, x^1)$, unlearning requests S_I and the corresponding tickets $\{t_z\}_{z \in S_I}$, return a if S_I is empty. Otherwise,

- If $a = \perp$, return $\perp$.
- Else, compute S_I^0 and S_I^1 as the set of 0-labeled and 1-labeled delete requests respectively. Similarly, define the tickets T_I^0 and T_I^1 as the tickets for the 0-labeled and 1-labeled samples in I respectively.
- Compute the max 0-labeled sample in remaining dataset via $x^0 = \text{MAXVAL}.\mathsf{Unlearn}(S_I^0, x^0, T_I^0)$.
- Compute the min1-labeled sample in remaining dataset via $x^1 = \text{MINVAL}.\mathsf{Unlearn}(S_I^1, x^1, T_I^1)$.
- If $x^0 = \perp$ or $x^1 = \perp$, return $\perp$.
- Else if $x^0 < x^1$, return $\perp$. Else, return $\top$.

Since, the MINVAL and MAXVAL primitives compute the minimum 1-labeled x^1, and the maximum 0-labeled samples x^0. Additionally, since realizability testing w.r.t. $\mathcal{H}_{\mathrm{th}}$ is equivalent to checking whether $x^0 < x^1$, one can verify that the above algorithm is correct. Since the MINVAL and MAXVAL primitives can be executed using a TiLU scheme with space complexity $(O(\log|\mathcal{X}|), O(\log|\mathcal{X}| + \log\alpha^{-1}(n)))$, the above TiLu scheme for realizability testing also has space complexity $(O(\log|\mathcal{X}|), O(\log|\mathcal{X}| + \log\alpha^{-1}(n)))$.

■