Robust Learning-Augmented Dictionaries

Ali Zeynali [1] **Shahin Kamali** [2] **Mohammad Hajiesmaili** [1]

Abstract

We present the first learning-augmented data structure for implementing dictionaries with optimal consistency and robustness. Our data structure, named `RobustSL`, is a skip list augmented by predictions of access frequencies of elements in a data sequence. With proper predictions, `RobustSL` has optimal consistency (achieves static optimality). At the same time, it maintains a logarithmic running time for each operation, ensuring optimal robustness, even if predictions are generated adversarially. Therefore, `RobustSL` has all the advantages of the recent learning-augmented data structures of Lin, Luo, and Woodruff (ICML 2022) and Cao et al. (arXiv 2023), while providing robustness guarantees that are absent in the previous work. Numerical experiments show that `RobustSL` outperforms alternative data structures using both synthetic and real datasets.

1. Introduction

Dictionaries are one of the most studied abstract data types with a wide range of applications, from database indexing (Bayer & McCreight, 1972) to scheduling in operating systems (Rubini & Corbet, 2001). In the dictionary problem, the goal is to maintain a set of n *items*, each represented with a *key*, so as to minimize the total *cost* of processing an online data sequence of operations, each involving access, insertion, deletion, or other operations such as successor and range queries. Here, the cost refers to the number of comparisons made for all operations, and *amortized cost* is the average cost of a single operation over the input sequence.

There are several data structures to implement dictionaries. Hash tables are efficient and practical data structures that are useful in many applications. However, hash tables do not efficiently support operations like successor and rank/select queries (Lin et al., 2022). In comparison, binary search trees (BST) and skip lists keep items sorted and thus support queries that involve ordering items with minimal augmentation. Classic BSTs such as red-black trees, AVL trees, treaps, and classic skip lists support dictionary operations in $\mathcal{O}(\log n)$ time, which is optimal for a *single* operation. For a *sequence* of m operations in a data stream, however, these structures are sub-optimal as they do not react to the access patterns in the input. For example, an input may be formed by m requests to a leaf i of a balanced BST, giving it a total cost of $\Theta(m \log n)$, while a solution that first moves i to the root has a cost of $\Theta(m + \log n)$. The same argument can be made by making requests to a key replicated once at the deepest level of a skip list. Splay trees (Sleator & Tarjan, 1985) are self-adjusting BSTs that move each accessed item to the tree's root via a splay operation. Splay trees are *statically optimal*, meaning their cost is proportional to an optimal data structure that does not self-adjust. Nevertheless, as pointed out by Lin et al. (2022) and Cao et al. (2023), the constant multiplicative overhead involved in pointer updates in the splay operations makes them impractical in many applications.

In recent years, there has been an increasing interest in augmenting algorithms that work on data streams with machine-learned predictions. The objective is to design solutions that provide guarantees with respect to *consistency*, the performance measure when predictions are accurate, and *robustness*, the performance measure when predictions are adversarial. For dictionaries, Lin et al. (2022) presented an augmented treap data structure that uses frequency predictions. In a classic treap, each item is assigned a random priority that defines its location in the tree. In the work of Lin et al. (2022), these random priorities are replaced with machine-learned frequency predictions. Under the "random-order rank" assumption, which implies all keys have the same chance of being the i'th frequently asked key (for any $i \leq n$), the resulting data structure offers static optimality with accurate predictions (is optimally consistent) and is robust in the sense that the expected cost of all operations in $\mathcal{O}(m \log n)$. When the ranks are not random, however, the resulting data structure is not consistent nor robust (§2.3, Proposition 2.2). Cao et al. (2023) have

[1]Manning College of Information & Computer Sciences, University of Massachusetts Amherst, USA [2]Department of Electrical Engineering & Computer Science, York University, Canada . Correspondence to: Ali Zeynali <azeynali@umass.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

recently proposed an alternative priority assignment that results in a learning-augmented treap (or learning-augmented B-Tree with priorities) that is statically optimal with accurate predictions without the random-order rank assumption. Nevertheless, we show that their structure is also vulnerable when predictions are adversarial and thus is not robust (§2.3, Proposition 2.3).

Contributions. This work introduces `RobustSL`, a learning-augmented data structure that utilizes frequency predictions and achieves optimal consistency and robustness. Our primary contributions are delineated as follows:

- In § 3.1, we presented `RobustSL`, a skip list that replicates each item, given its predicted frequency, based on two factors. The first factor is deterministic and is calculated based on the predicted frequency of the item using a classification approach that ensures items with similar predicted frequencies receive a comparable number of replicas (number of times a particular element appears at different levels) within the skip list. In particular, including this factor guarantees that items with higher predicted frequencies obtain more expected number of replicas, thereby optimizing the structure's performance. The second factor is a randomized factor added to the deterministic one. The intuition behind this factor is similar to classic skip lists: adding a random "noise" to the number of replicas to ensure efficient list navigation when searching for an item. The deterministic replication process ensures consistency, whereas the stochastic mechanism fortifies the robustness of `RobustSL`.
- We then analyze the consistency and robustness of `RobustSL`. Theorem 3.3 establishes consistency of the `RobustSL` by revealing that under perfect frequency predictions, the anticipated access cost for an item i within `RobustSL` is proportional to the logarithm of its predicted frequency, which is a consequence of the deterministic replication mechanism. This relationship confirms that the access cost for an input sequence is a constant factor away from the entropy of the input, thus proving the optimal consistency of `RobustSL`. Additionally, our analysis demonstrates that the maximum cost of access to any item in `RobustSL` remains within $\mathcal{O}(\log n)$, substantiated by the stochastic replication mechanism), solidifying the structure's assured optimal robustness (Theorem 3.4). We further show the number of comparisons made by `RobustSL` increases smoothly as a function of error, measured by the *cross entropy* between predicted and actual frequency distributions (Theorem 3.5).
- In §4, we conduct comprehensive experimental comparisons between `RobustSL` and other dictionary data structures using synthetic and real datasets. Our experimental findings align closely with theoretical analysis, demonstrating that data structures such as learned treaps exhibit satisfactory performance for inputs with highly accurate predictability. However, their performance is significantly worse when predictions are erroneous. In comparison, `RobustSL` achieves comparable performance with accurate predictions and remains robust even for highly erroneous predictions.

2. Preliminaries

We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$ and assume keys in the dictionary come from universe $[n]$. We use m to denote the number of operations in the input stream. We also let f_i denote the frequency of operations involving key $i \in [n]$ in the input, and $\hat{f}_i$ denote the predicted frequency for queries to i ($\sum_i f_i = \sum_i \hat{f}_i = 1$). We let $\boldsymbol{f}$, and $\hat{\boldsymbol{f}}$ denote the vectors of actual and predicted frequencies. Let $e_i \in [n]$ be the item with *rank* i; that is, the i'th most frequently-accessed item in the input. Finally, we use $\texttt{Operation}_D(i)$ to denote the number of comparisons for applying an operation (the "`Operation`") involving key $i \in [n]$ in a data structure D. For example, $\texttt{Search}_D(i)$ is the number of comparisons for accessing i in D.

2.1. Dictionaries and Optimality

Consider an online stream of operations concerning a set of n items, forming a dictionary. The operations mainly require accessing, inserting, and deleting items, while secondary operations such as rank, select, range-query, successor, and predecessor may need to be supported. *Comparison-based* data structures, such as BSTs and skip lists, keep data in order and thus support all operations without relying on restrictive assumptions on the input distribution. The cost of these data structures for a given input stream can be measured by the number of comparisons they make for all operations in the input; other costs, such as pointer updates, are proportional to the number of comparisons. Most comparison-based data structures, such as balanced BSTs and skip lists, do not change structure after access operations. *Self-adjusting* data structures such as splay trees (Sleator & Tarjan, 1985), on the other hand, require extra comparisons for self-adjusting and thus have a constant-factor overhead, which is not desirable in many applications (Lin et al., 2022).

Mehlhorn (1975) showed that the number of comparisons made by any comparison-based data structure that does not self-adjust, for the input of m operations and frequency distribution $\boldsymbol{f}$, is at least $mH(\boldsymbol{f})/\sqrt{3}$, where $H(\boldsymbol{f})$ is the *entropy* of $\boldsymbol{f}$ defined as $H(\boldsymbol{f}) = -\sum_{i\in[n]} f_i \log(f_i)$. Therefore, one can use entropy as a reference point for measuring

the consistency of learning-augmented structures. In particular, we say that a structure is *statically optimal* if its cost for any instance $\mathcal{I}$ is $\mathcal{O}(H(\boldsymbol{f}))$. In particular, the weight-balanced BST of Knuth (1971) and its approximation by Mehlhorn (1975) are statically optimal.

To measure robustness, i.e., performance under adversarial error, we note that, regardless of the structure of comparison-based data structure, there are access requests that require at least $\lceil \log n \rceil$ per access due to the inherent nature of binary comparisons. Therefore, when predictions are adversarial, there are worst-case input sequences in which every access request takes at least $\lceil \log n \rceil$ comparisons ($\Omega(m \log n)$ cost for a sequence of m operations). From the above discussions, we define our measures of optimality as follows.

Definition 2.1 (Optimal consistency and robustness)**.** An instance $\mathcal{I} = (\sigma, \boldsymbol{f}, \hat{\boldsymbol{f}})$ of the dictionary problem with prediction includes a sequence σ of m operations on n keys, an unknown vector $\boldsymbol{f}$ specifying frequency (probability) of keys in σ, and a known vector $\hat{\boldsymbol{f}}$ of predicted frequencies. A learning-augmented data structure has consistency c iff its total number of comparison is $c \cdot H(\boldsymbol{f})$ for any input instance with $\boldsymbol{f} = \hat{\boldsymbol{f}}$. In particular, it is *optimally consistent*, or statically optimal, if its consistency is $\mathcal{O}(1)$. Similarly, a learning-augmented data structure is r-robust iff its total cost is at most $r.m$ for any instance. In particular, it is *optimally robust* iff it has robustness $\mathcal{O}(\log n)$ for a dictionary with n items.

Consistency and robustness are not always inherently conflicting attributes, yet achieving one often necessitates compromising the other, as observed in prior research (Angelopoulos et al., 2020; 2022; Zeynali et al., 2021; Li et al., 2022; Sun et al., 2021). Attempts to attain optimal consistency have often resulted in a trade-off that undermines robustness and vice versa. Within the literature (Knuth, 1971; Cao et al., 2023; Lin et al., 2022), several data structures have been proposed with an emphasis on optimizing consistency, showcasing exceptional performance under ideal predictive conditions. However, this pursuit of high consistency tends to render these structures less robust when faced with adversarial predictions. Conversely, other data structures, such as balanced BSTs and AVL tree, prioritize bolstering their robustness, particularly under worst-case scenarios, thereby offering enhanced resilience. In light of these results, one might ask whether it is possible to get the best of the two worlds: optimal consistency and robustness at the same time. We answer this question in the affirmative in this paper.

2.2. Treaps and Skip Lists

A treap is a binary search tree where items have an additional field which is its priority. In addition to the binary search tree property, a treap follows the heap property. In a classic *random treap* (Seidel & Aragon, 1996), priorities are assigned randomly, allowing them to support dictionary operations in expected $\mathcal{O}(\log n)$ time, i.e., they are optimally robust. On the other hand, random treaps are not optimally consistent due to a lack of information about access frequencies. When frequency predictions are available, it is natural to define priorities based on frequencies rather than randomly. These ideas were explored in (Lin et al., 2022; Cao et al., 2023) to define learning-augmented treaps (see §2.3).

Skip lists, introduced by Pugh (1990), are randomized data structures that provide all functionalities of balanced BSTs on expectation. A skip list is a collection of sorted linked lists that appear in *levels*, where all dictionary items are present in the lowest level, and a subset of items at each level are selected randomly and independently to be replicated in the above layer. The random decisions follow a geometric distribution, ensuring an expected $\mathcal{O}(1)$ replicas per key. Each replica has a "right pointer" to the next item in its linked-list, and a "bottom pointer" to the replica of the same item in the level below. The top list is said to have level 1, and the level of any other list is the level of its top list plus 1. The "height" of skip lists, is defined as the number of levels in it, and "depth" of a specific item, is the least level number at which the item appears.

Previous work has established a tight relationship between skip lists and trees. For example, a skip list can simulate various forms of multiway balanced search trees (MWBSTs) such as B-trees, and (a,b)-trees (Munro et al., 1992; Mehlhorn & Näher, 1992; Bagchi et al., 2005). On the other hand, *Skip trees* (Messeguer, 1997) are MWBSTs that certify one-to-one mapping between MWBSTs and skiplists. It is also possible to represent any skip list with a BST (Dean & Jones, 2007; Bose et al., 2008).

2.3. Learning-augmented Treaps

In the learning-augmented treaps proposed by Lin et al. (2022), priorities are not random but instead predicted frequencies. Under a *random-order rank assumption*, which requires that items' ranks to be a random permutation of $[n]$, these treaps are optimally-consistent and have bounded robustness. However, these treaps are not optimally consistent nor robust when the random-order rank assumption does not hold. In particular, when the predicted frequency distribution is highly skewed, the resulting treap may resemble a linear list, which is clearly neither robust nor consistent, as shown in the following proposition.

Proposition 2.2 (Appendix §A.1)**.** *Without random-order rank assumption, the consistency of learning-augmented treap of Lin et al. (2022) for dictionaries of size n is at least $\Omega(n/\log n)$, and its robustness is n.*

Lin et al. (2022) present a method for relaxing the random-rank assumption by bijecting each key in the dictionary to a random key in a secondary dictionary maintained by a treap based on predicted frequencies. This simple solution, however, shuffles the keys, and the resulting tree is not ordered. Therefore, one cannot efficiently answer secondary dictionary operations such as predecessor and successor operations using these shuffled treaps.

Recently, Cao et al. (2023) introduced a different learning-augmented treap (or learning-augmented B-Trees with priorities), in which a random element is introduced in priority assignment. Precisely, the priority of the node with key i is defined as $\texttt{pri}_i = \delta_i - \lfloor \log_2 \log_2 1/\hat{f}_i \rfloor$, $\delta_i \sim U(0,1)$, where $U(0,1)$ is the uniform distribution over interval $[0,1]$. The total number of comparisons is proved to be $\mathcal{O}(m \cdot \texttt{Ent}(\boldsymbol{f}, \hat{\boldsymbol{f}}))$, where $\texttt{Ent}(\boldsymbol{f}, \hat{\boldsymbol{f}})$ is the *cross entropy* between $\boldsymbol{f}$ and $\hat{\boldsymbol{f}}$ defined as $\texttt{Ent}(\boldsymbol{f}, \hat{\boldsymbol{f}}) = -\sum_{i \in [n]} f_i \log \hat{f}_i$. When $f_i = \hat{f}_i$, it holds that $\texttt{Ent}(\boldsymbol{f}, \hat{\boldsymbol{f}}) = H(\boldsymbol{f})$, and the total number of comparisons will be $\mathcal{O}(mH(\boldsymbol{f}))$, ensuring optimal consistency for these treaps. However, when predictions are adversarial, these treaps are not robust, as shown in the proposition below.

Proposition 2.3 (Appendix §A.2)**.** *The learning-augmented treaps of (Cao et al., 2023) have optimal consistency (are statically optimal), but their robustness is $\mathcal{O}(n)$.*

Intuitively, regardless of the amount of randomness (noise) added to the priorities, an adversary can skew the predicted frequency distribution to negate the added random noise. For that, it suffices to define $\hat{f}_i$'s in a way to ensure $\hat{f}_i \geq \hat{f}_{i-1} + u$, where u is the upper bound for the random variable added to the priority.

3. Consistent and Robust Dictionaries

This section presents our main results on learning-augmented dictionary data structures: a skip list that achieves optimal consistency (statically optimal) when predictions are correct and offers robustness of $\mathcal{O}(\log n)$ when predictions are adversarial.

If one wants to maintain a *static dictionary*, without the support of insertions and deletions, it is rather easy to achieve a consistent and robust data structure as follows. Provided with predicted frequencies $\hat{\boldsymbol{f}}$, first, build the static optimal tree of Knuth (1971) in $\mathcal{O}(n^2)$; call the resulting tree *optimistic tree* T_o. Also, form a balanced BST T_p, referred to as *pessimistic tree*. To access an item with key i, we simultaneously search for i in T_o and T_p. When one tree finds the item, we immediately stop the searching process of another tree. The number of comparisons would be $2\min(\text{Search}_{T_o}(i), \text{Search}_{T_p}(i))$, ensuring both static optimality and optimal robustness of $O(\log n)$.

However, this simple data structure does not support insertion and deletion queries, as Knuth's static optimal tree lacks support for these operations. In practice, the predicted frequencies get updated frequently, and such updates are implemented by deletion and insertion, necessitating a *dynamic dictionary that supports insertions and deletions*. Moreover, the time complexity of constructing the optimistic tree is expensive, rendering it impractical for widespread use.

In what follows, we first present `RobustSL` in Section 3.1. In Section 3.2, we show that `RobustSL` achieves optimal consistency and robustness. Last, in Section 3.3, we discuss extensions, such as how `RobustSL` can be augmented to efficiently support secondary operations like the predecessor, successor, rank, select, and range queries.

3.1. `RobustSL`, Consistent and Robust Skip list

`RobustSL` is a skip list, and many of its functionalities are similar to a regular skip list. In particular, upon insertion of an item with key i, it replicates the item in a certain number of levels (each associated with a linked list), starting at the lowest level. Unlike a regular skip list, where the replication strategy is purely randomized, `RobustSL` involves predicted frequencies in its replication strategy. As we will show, the number of replicas for an item with key i in `RobustSL` is a function of its predicted frequency and a geometric random variable. The key to achieving optimal consistency and robustness in `RobustSL` lies in the precise classification of items based on their predicted frequencies. The goal is to assign more replicas (lower "depths") to items with higher predicted frequencies, while items with similar predicted frequencies share the same expected depth, ultimately reinforcing the consistency of `RobustSL`. Importantly, `RobustSL` ensures that the maximum number of comparisons made while searching any item is at most $\mathcal{O}(\log n)$, ensuring optimal robustness.

Before presenting `RobustSL` formally, we present some intuitions behind it via a simpler data structure that illustrates the main ideas behind `RobustSL`.

High-level intuitions and ideas. Let's assume the number of items in the dictionary, n, is fixed (this assumption will be relaxed later in Section 3.3). We classify items such that items with predicted frequency larger than 1/2 belong to the first class (index 0), those with predicted frequency in $(1/4, 1/2]$ belong to class index 1, and more generally, items with predicted frequency in $(1/2^{2^c}, 1/2^{2^{c-1}}]$ belong to class c. We further limit the number of classes into $K = \lceil \log \log n \rceil$; that is, items with frequency smaller than $1/n$ belong to class K. Maintain a separate skip list for items of each class, and, to search for any item with key i, first examine the skip list maintained for the first class, and in case of not finding i, we examine the skip list for classes

$2, 3, \ldots, K$ in the same order.

We show that searching for an item in class $c < K$ takes no more than $\alpha \cdot 2^{c+1}$ comparisons on expectation for some constant α. To see that, let N_c denote the number of items in class c. Observe that at most $2^{(2^{c+1})}$ items belong to class c (i.e., $N_c \leq 2^{(2^{c+1})}$); otherwise, the total predicted frequencies for items in class c exceeds 1. Therefore, searching for an item in the skip list of class c takes the expected time of $\alpha \log N_c \leq \alpha 2^{c+1}$. Now, if i is assigned to class index c, the total expected number of comparisons for finding i would be $\sum_{c' \leq c} \alpha 2^{c'+1} < \alpha 2^{c+2}$. The robustness of the data structure is direct from the fact that $c < K = \lceil \log \log n \rceil$, and thus searching any item takes at most $4\alpha 2^K = O(\log n)$. For consistency, when predictions are accurate, we note that if an item with key i is assigned to class c, then it holds that $f_i = \hat{f}_i \leq 2^{-2^c}$, which is direct from the way classes are defined. Thus, the expected number of comparisons for finding i would be $\alpha \cdot 2^{c+2} \leq -4\alpha \cdot \log \hat{f}_i$. We can conclude that searching for i takes $O(-\log f_i)$, proving our algorithm's consistency.

`RobustSL` improves the above data structure in a few ways. First, it maintains a *single* skip list for all classes, which is necessary for efficient handling of frequency updates: when an item's frequency is updated, it is desirable to update the number of replicas in a single skip list rather than removing them from one skip list and adding to another. To maintain a single skip list, we ensure that items of class c form a "layer" above those of class $c+1$; this is achieved via defining a "class-base" for each class, ensuring a minimum number of replicas for class c. Second, instead of using powers of $1/2$ in the classification, `RobustSL` uses a parameter θ, which is set to optimize the constant for static optimality.

Algorithmic details of `RobustSL`. Similarly to the data structure described above, `RobustSL` classifies items based on their predicted frequencies and employs a two-step process to select the number of replicas for each item. Firstly, it replicates any item in class c a minimum of $\mathcal{D}(K) - \mathcal{D}(c)$ times , where $\mathcal{D}(c)$ is an increasing function of c, indicating the maximum possible depth of items in class c. Given this definition, $\mathcal{D}(K) - \mathcal{D}(c)$, called `class-base(c)`, determines the minimum number of replicas for items in that class. Intuitively, items in class 0, which are predicted to appear more in the input, have a higher class-base, meaning that they are replicated more than other classes; as the indices of classes increase, their class-base decreases, implying less replication. See Figure 1 for an illustration.

In addition to replicas specified by the class-base, more replications are introduced for each item, guided by a geometric distribution with a parameter of $1-p$, where $p < 1$ is a constant value. In the next section, we present the consistency

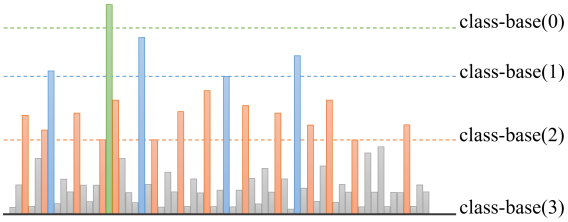

Figure 1. Structure of `RobustSL` having four classes. Items in lower-indexed classes replicate more. The replication of items within a class is achieved through a stochastic process.

and robustness of `RobustSL` as a function of p, and show how this parameter impacts those metrics. This stochastic process for adjusting the number of replicas ensures that the expected search cost within a class is logarithmic to the number of items in that class. The following sections explore the classification and replication processes within `RobustSL` in detail.

Classification approach. The classification of items relies on their predicted frequencies. An item i with a predicted frequency $\hat{f}_i$ is assigned to class $c_i (\geq 0)$ as follows. First, items with predicted frequency $\geq \theta$ belong to class 0. Other items belong to class $c_i \geq 1$, if their predicted frequency satisfies the condition:

$$\theta^{2^{c_i}} \leq \max(\hat{f}_i, n^{\log(p)/2}) < \theta^{2^{c_i - 1}}, \tag{1}$$

which gives

$$c_i = \lceil \log \left(\frac{-1}{\log \theta} \min \left(-\log \hat{f}_i, \frac{-\log p \log n}{2} \right) \right) \rceil,$$

where $\theta \in (0, 1)$ is an algorithm parameter. In particular, the largest value of c_i is realized for items with small predicted frequency, where $-\log \hat{f}_i \leq -\log p \log(n)/2$. These items (if exist) belong to the class index denoted by K, where

$$K = 1 + \lceil \log \log n - \log(2 \log \theta / \log p) \rceil.$$

Intuitively speaking, parameter θ determines the predicted frequency range for items in the same class. A smaller value of θ widens the range, which implies fewer classes, while higher θ narrows it, resulting in more classes. Regardless, the number of classes is bounded by $K + 1 = \Theta(\log \log n)$.

Number of replicas. To determine the number of replicas for an item upon its insertion, `RobustSL` first calculates the class-base of the item, which is deterministically defined based on the class of item, c, and is calculated as $\mathcal{D}(K) - \mathcal{D}(c)$. We recursively define $\mathcal{D}(c)$ as follows. First, for $c = 0$, we let $\mathcal{D}(0) = \lceil \log \theta / \log p \rceil$, and for any $c > 0$, we define:

$$\mathcal{D}(c) = \mathcal{D}(c-1) + \lceil \frac{\log \theta}{\log p} 2^c \rceil. \tag{2}$$

Intuitively, $\mathcal{D}(c)$ represents the maximum depth of items belonging to class c, and the above definition ensures that

items in classes larger than c appear at least $2^c \log\theta/\log p$ levels deeper than items of class c, on expectation. The number of replicas for an item i in class c_i is calculated as:

$$h_i = \mathcal{D}(K) - \mathcal{D}(c) + \mathcal{X}(p), \tag{3}$$

where $\mathcal{X}(p)$ is a random variable following a geometric distribution with parameter $1-p$. For example, items of class K (if there is any), which are the ones with the smallest predicted frequencies, are replicated only based on stochastic replications, $\mathcal{X}(p)$, ensuring that they are at the deepest levels of the skip list. As another observation, for any $c < c'$, items of class c are replicated in $D(c') - D(c)$ more levels, on expectation, than items of class c'. This is consistent with the higher predicted frequencies for items of class c. Finally, note that items that belong to the same class all have the same expected number of replicas, ensuring that the data structure resembles a regular skip list for items within the same class.

3.2. Theoretical Analysis of `RobustSL`

In what follows, we analyze `RobustSL`, providing upper bounds on the number of comparisons made when accessing any item. In particular, we establish the optimal consistency and robustness of `RobustSL`. Let $c \in [0, K]$ be any class defined by `RobustSL`; we use N_c to denote the number of items belonging to class c. If $c \leq K-1$, then for any item i in class c, it holds that $\hat{f}_i \geq \theta^{2^c}$. Consequently, it follows that $N_c \cdot \theta^{2^c} \leq 1$, as violating this condition would contradict the requirement that predicted frequencies form a probability distribution ($\sum_i \hat{f}_i = 1$). Thus, we deduce that $N_c \leq \theta^{-2^c}$, leading to the inequality $\log N_c \leq -2^c \log\theta$.

The following lemma presents an analysis of the classification and replication mechanisms within `RobustSL`, illuminating that beyond the inherent relationship between higher predicted frequencies and greater heights, items belonging to distinct classes are anticipated to possess varying numbers of replications.

Lemma 3.1 (Appendix §A.3)**.** *For any class index $c > 0$, the expected number of items of class c that are replicated at least `class-base(c-1)` times is less than 1.*

This lemma indicates that, while searching for a key i categorized within a specific class, c_i, the expected number of comparisons conducted with items in higher-indexed classes, $c_j > c_i$, is significantly lower than the expected comparisons with items in class $c \leq c_i$. This property helps `RobustSL` to access more frequent items with a lesser number of comparisons. Using this lemma, we can provide an upper bound from the number of comparisons made while searching for an item i.

Lemma 3.2 (Appendix §A.4)**.** *The expected number of comparisons made while searching an item i in class $c_i < K$ is at most $\frac{4}{p\log p}\log(\hat{f}_i)$. For an item i in class $c_i = K$, the expected number of comparisons is at most $\frac{4\log\theta - 1}{p\log p}(\log n) + \mathcal{O}(1)$.*

In the following, we establish the consistency of `RobustSL`, which follows from Lemma 3.2, noting the number of comparisons for an item i of frequency $f_i = \hat{f}_i$ is proportional to $\log(f_i)$ for all classes, including class K.

Theorem 3.3 (Appendix §A.5)**.** *The consistency of `RobustSL` can be established as $\frac{-4}{p\log(p)} = \mathcal{O}(1)$, ensuring the static optimality for `RobustSL`.*

The following lemma is direct from Lemma 3.2, by noting that the maximum number of comparisons made while searching for items belong to class K.

Theorem 3.4 (Appendix §A.6)**.** *The maximum cost of searching for items within `RobustSL` is $\mathcal{O}(\log n)$, which makes `RobustSL` optimally-robust.*

The above two theorems establish that `RobustSL` is optimally consistent and robust. We can further leverage Lemma 3.2 to bound the expected number of comparisons of `RobustSL` as a function of prediction error, where the error is quantified by the cross entropy between the predicted and actual frequencies.

Theorem 3.5 (Appendix §A.7)**.** *The expected number of comparisons made within `RobustSL` while searching for items with actual frequency of $\boldsymbol{f}$ and predicted frequency of $\hat{\boldsymbol{f}}$ is $\mathcal{O}(\mathtt{Ent}(\boldsymbol{f}, \hat{\boldsymbol{f}}))$.*

3.3. Discussion and Extensions

Dynamics of `RobustSL`. The results in the previous section assume a fixed value of n. The actual number of items after t^{th} query, denoted by n_t, however, is clearly impacted by the insertions and deletions. We explain how to maintain the described consistency and robustness, using a value of n, which approximates n_t and represents the parameter utilized by `RobustSL` for classification (as depicted in Equation (1)). We maintain a value of n that is always larger than n_t. Initially, we set $n = 4$. After any insertion operation, if n_t becomes equal to n, we update $n \leftarrow n^2$. After a deletion operation, if $n_t = n^{1/4}$, we adjust $n \leftarrow \max(4, \sqrt{n})$. Whenever n is updated, we reconstruct `RobustSL`.

This approach does not compromise the consistency of `RobustSL`, as $n_t \leq n$, preserving the analysis outlined in Theorem 3.3. Additionally, n remains bounded by $n < n_t^4$ for any $n_t > 1$, ensuring $\log n = \mathcal{O}(\log n_t)$ consistently. Consequently, the robustness of `RobustSL` maintains at $\log n = \mathcal{O}(\log n_t)$.

By employing this technique, the amortized cost of insertion or deletion operations remains $\mathcal{O}(\log n)$ (finding the index of a key in the sorted ordering of all key values requires $\Omega(\log n)$ number of operations), as updates to

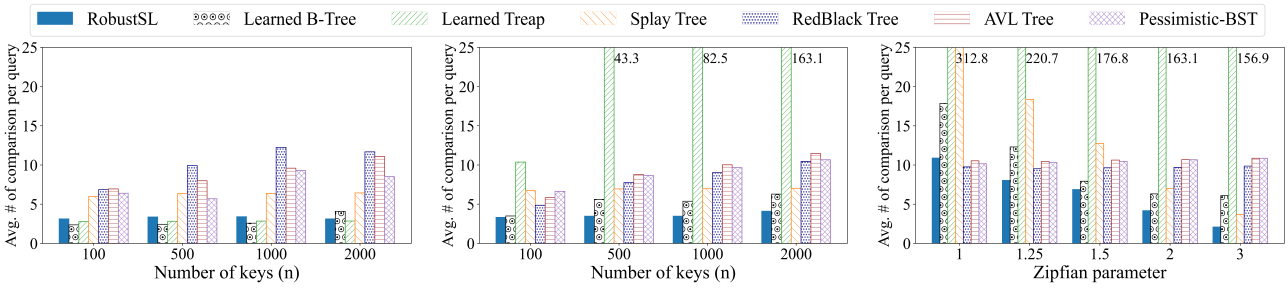

Figure 2. Average number of comparisons per query of `RobustSL` and baseline data structures for dynamic evaluations under (left) random frequency ordering with perfect predictions, (center) adversary frequency ordering with noisy predictions, and (right) adversary frequency ordering with different value of Zipfian parameter. The performance of learned treap is highly impacted by the prediction error, while `RobustSL` shows its robustness against noisy predictions. To improve visibility, the y-axis range in the right figure is limited to 25, with actual values displayed next to bars exceeding this threshold for clarity.

`RobustSL` occur only after at least n^2 consecutive insertions or deletions. Note that the maximum cost for entirely reconstructing `RobustSL`, as well as its memory requirement, is capped at $\mathcal{O}(n \log n)$, given its maximum depth as $\mathcal{O}(\log n)$, is capped at $\mathcal{O}(n \log n)$.

Secondary queries. The order structure of the skip list allows efficient answering of secondary queries with little augmentation. First, we augment the deepest level of `RobustSL` to make it doubly linked. We also add a pointer from any node x, $deep(x)$, to the node with the same key as x at the deepest level. This allows implementing `predecessor`(i)/`successor`(i) (keys before/after i in the sorted order) by searching for i, finding it at some node x, following $deep(x)$, and probing a left/right pointer; the time complexity will be similar to the search. Similarly, `range`(i, j) (reporting all keys k s.t. $i \leq k \leq j$) can be done by searching for i and following pointers in the deepest level. For other secondary queries, let $pred(x)$ be the node before x at the same level. We let x have an extra integer field $s(x)$ that specifies the number of keys that are "skipped" by following the right pointer from $pred(x)$ to x, that is, the number of keys in the dictionary between the key of $pred(x)$ and i. These augmentations allow efficient answering of `rank`(i) (the index of i in a sorted ordering of keys) by searching for i and summing the values of $s(x)$ over nodes on the search path. Similarly, `select`(t) (the t'th smallest key) can be done in $\mathcal{O}(\log n)$, summing over values $s(x)$ by following the right pointers to reach t.

Proposition 3.6. *It is possible to augment `RobustSL` to answer `predecessor`(i), `successor`(i), `rank`(i), all in time proportional to `Search`(i), `range`(i, j) in time proportional to `Search`$(i) + z$, where z is the output size, and `select`(t) in $\mathcal{O}(\log n)$.*

4. Experiments

In this section, we evaluate the performance of `RobustSL` in both static and dynamic settings and compare the performance with alternatives. Our goal is to investigate the robustness and consistency of `RobustSL` under perfect and adversarial predictions.

4.1. Experimental Setup and Overview

We compare `RobustSL` with several alternative data structures, namely AVL trees, red-black trees, splay trees, balanced BST (pessimistic BST), learning-augmented treaps of Lin et al. (2022), and learning-augmented treaps/B-trees of Cao et al. (2023). We thoroughly evaluate the robustness and consistency of `RobustSL` in static and dynamic settings by considering both perfect predictions and adversarial instances. Specifically, we conduct our experiments in two categories: (1) dynamic random orders with perfect predictions, and (2) dynamic adversarial orders with noisy predictions. In addition, we provide further experiments concerning static dictionaries in Appendix B. In the random order experiments, the frequency ranking of items are randomly selected concerning the key value ranking of items. In addition, we consider a perfect prediction in random order experiments to compare the *consistency* of `RobustSL` and baseline algorithms. On the other hand, the adversarial ranking is used to evaluate the *robustness* of `RobustSL` as compared to alternatives and includes perfect and noisy predictions. For each category, we conduct a series of 100 trials and report the average number of comparisons for a search query of `RobustSL` and baseline data structures. More details on the prediction and error model are given in Appendix§ B. We select $\theta = 0.05$ in `RobustSL` for our experiments since it leads to slightly better performance, even though `RobustSL` is robust to the selection of this parameter (as shown in Figure 5 in AppendixB.2). In addition, we select $p = 0.368$ that minimizes the consistency value presented in Theorem 3.3. Last, we use the following two

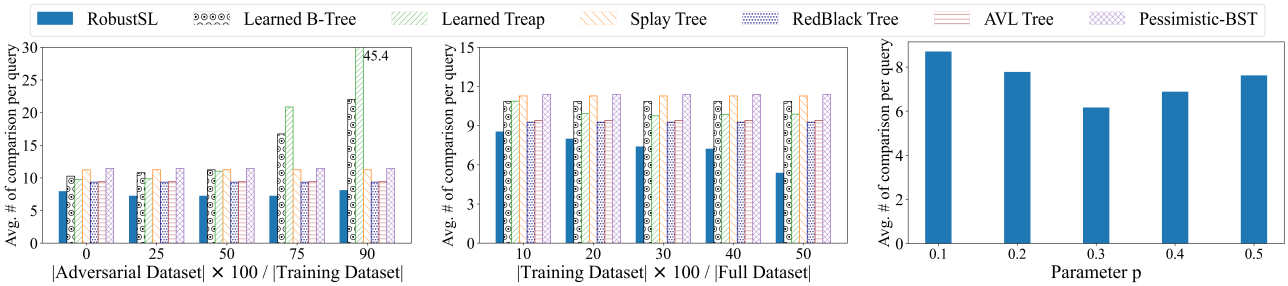

Figure 3. Average number of comparisons per query of RobustSL and the baseline data structures. The evaluation is conducted by varying three factors: (left) sizes of the adversarial dataset, (center) sizes of the training dataset, and (right) value of p in RobustSL. Notably, the performance of learning-augmented treaps and learning-augmented B-Trees demonstrates a significant increase when predicted frequencies are adversarial.

categories of synthetic and real datasets.

Synthetic dataset. We conducted experiments to evaluate the performance of RobustSL and baseline data structures using synthetic datasets of varying sizes. Specifically, we selected the number of unique keys, n, from the set of values $[100, 500, 1000, 2000]$. For each value of n, we generated $m = 100,000$ search queries with each item appearing according to the Zipfian distribution with a parameter of 2. In addition, we test the performance of RobustSL and other data structures against Zipfian distribution with different parameters. For random order experiments and experiments on testing different values of Zipfian distribution, we fix the number of items to 2000. Our results demonstrate that the performance of learning-augmented treaps is much worse against the adversary frequency ordering using low-accuracy predicted frequencies. For a fair comparison between the performance of RobustSL and the baselines data structures, in experiments with adversarial frequency predictions, we select the number of trials with high-accuracy predictions up to 9 times of low-accuracy prediction trials. This means that in the experiments with "adversarial" frequency predictions, 90 trials used high-accuracy predictions, and a 10 trial was designed using low-accuracy predictions (see Appendix §B for details).

BBC news article dataset. We also use the BBC news article dataset (Kushwaha, 2023) to evaluate the performance of RobustSL and other baseline data structures in responding to news article queries (Chen et al., 2022; Kostakos, 2020). In these experiments, we select a fraction of the entire dataset to predict item frequencies (training dataset) in the remaining portion (test dataset). To evaluate data structure performance under noisy conditions, we artificially generated additional articles using an adversarial approach. These adversarially-generated articles were designed to align the ranking of frequencies with keys. In addition, we test the impact of parameters p and θ on the performance of RobustSL against this dataset. We randomly select 40% of the entire data as the training dataset, 25% (of the training dataset) as the size of the adversary dataset, $\theta = 0.05$, and $p = 0.368$ (as used in synthetic experiments) and conducted experiments by varying one factor during each experiment while other factors remained fixed. Our analysis considers the top 5500 items with the highest frequencies for dictionary generation, and search queries were exclusively conducted on these items.

4.2. Experimental Results

Synthetic results. Figure 2 presents the results of our experiments on dynamic structures using synthetic data, showcasing the average number of comparisons per query for both RobustSL and baseline data structures under random and adversary settings. While the performance of learning-augmented treaps is notably impacted by error rates, RobustSL demonstrates consistent and robust performance, validating its theoretical resilience. Specifically, when searching a dictionary of size 2000, under random frequency ordering, the average number of comparisons of RobustSL is 9.5% more than of learning-augmented treaps under random frequency ordering while RobustSL achieves $40.0\times$ lower comparison per query when the predicted frequencies were adversarial. Notably, we repeated the adversarial experiments with a higher number of items (e.g., $n = 10000$); the average number of comparisons per query slightly increased for all dictionaries, while for learned treaps, it increased to 788, which confirms the vulnerability of these data structures to errors in prediction. Finally, we replicated the same experiment in a static setting, and since the results were very similar to those obtained in the dynamic setting, we have included the results and additional analysis on static experiments in Appendix §B.

Dataset results. Figure 3 illustrates the average number of comparisons per query for the compared data structures, varying size of adversarial and training datasets, and p in RobustSL. Results align with theoretical analyses

and synthetic dataset evaluations. Learned treaps perform well when the prediction error is low (zero-size adversarial dataset), but their performance significantly deteriorates under fully adversarial conditions (i.e., when the size of the adversarial dataset is comparable to the training dataset). In contrast, RobustSL demonstrates consistent performance and robustness throughout these experiments. When testing the impact of the training sample size, RobustSL shows significant improvement in its performance (smaller number of comparisons) with a larger training dataset, while this improvement for learned treaps was much lesser. Finally, testing the impact of p shows that the performance of RobustSL is optimized when p is around 0.3, supporting theoretical results. It is worth mentioning that our analysis reveals the impact of θ on the performance of RobustSL is negligible. The result of this experiment is given in Appendix §B.

5. Conclusion

In this paper, we presented RobustSL, a skip-list-based data structure that achieves optimality with quality predictions and stays robust with adversarial prediction. As a prospect for future exploration, adjusting RobustSL's structure to support the availability of partial information, particularly when frequency predictions are available only for a subset of items, presents an interesting avenue for future work. Whether the guarantees provided by RobustSL concerning consistency and robustness can be achieved with any learning-augmented BST is another open question for future study.

Acknowledgments

The work of Ali Zeynali and Mohammad Hajiesmaili was supported by the U.S. National Science Foundation (NSF) under grant numbers CCF-2325956, CAREER-2045641, CPS-2136199, CNS-2106299, and CNS-2102963. Shahin Kamali was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant number DGECR-2018-00059.

Impact Statement

This paper presents work whose goal is to advance the field of Trustworthy Machine Learning and Trustworthy data structures. There may be potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Angelopoulos, S., Dürr, C., Jin, S., Kamali, S., and Renault, M. P. Online Computation with Untrusted Advice. In *Proc. ITCS*, pp. 52:1–52:15, 2020.

Angelopoulos, S., Kamali, S., and Shadkami, K. Online Bin Packing with Predictions. In *Proc. IJCAI*, pp. 4574–4580. ijcai.org, 2022.

Bagchi, A., Buchsbaum, A. L., and Goodrich, M. T. Biased Skip Lists. *Algorithmica*, 42(1):31–48, 2005.

Bayer, R. and McCreight, E. M. Organization and Maintenance of Large Ordered Indices. *Acta Informatica*, 1: 173–189, 1972.

Bose, P., Douïeb, K., and Langerman, S. Dynamic optimality for skip lists and B-trees. In *Proc. SODA*, pp. 1106–1114. SIAM, 2008.

Cao, X., Chen, J., Chen, L., Lambert, C., Peng, R., and Sleator, D. Learning-Augmented B-Trees. 2023. doi: 10.48550/arXiv.2211.09251. URL https://doi.org/10.48550/arXiv.2211.09251.

Chen, X., Zeynali, A., Camargo, C., Flöck, F., Gaffney, D., Grabowicz, P., Hale, S., Jurgens, D., and Samory, M. SemEval-2022 task 8: Multilingual news article similarity. In *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*, pp. 1094–1106, Seattle, United States, July 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.semeval-1.155.

Dean, B. C. and Jones, Z. H. Exploring the duality between skip lists and binary search trees. In John, D. and Kerr, S. N. (eds.), *Proc. ACM Southeast Regional Conference*, pp. 395–399, 2007.

Knuth, D. E. Optimum Binary Search Trees. *Acta Informatica*, 1:14–25, 1971.

Kostakos, P. Strings and things: a semantic search engine for news quotes using named entity recognition. In *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 835–839. IEEE, 2020.

Kushwaha, S. BBCFull Text Document Classification. https://www.kaggle.com/datasets/shivamkushwaha/bbc-full-text-document-classification, 2023. Accessed: 2023-05-17.

Li, T., Yang, R., Qu, G., Shi, G., Yu, C., Wierman, A., and Low, S. H. Robustness and Consistency in Linear Quadratic Control with Untrusted Predictions. In *Proc. SIGMETRICS*, pp. 107–108. ACM, 2022.

Lin, H., Luo, T., and Woodruff, D. P. Learning Augmented Binary Search Trees. In *Proc. ICML*, volume 162, pp. 13431–13440. PMLR, 2022.

Mehlhorn, K. Nearly Optimal Binary Search Trees. *Acta Informatica*, 5:287–295, 1975.

Mehlhorn, K. and Näher, S. Algorithm Design and Software Libraries: Recent Developments in the LEDA Project. In *Proc. IFIP*, volume A-12, pp. 493–505. North-Holland, 1992.

Messeguer, X. Skip Trees, an Alternative Data Structure to Skip Lists in a Concurrent Approach. *RAIRO Theor. Informatics Appl.*, 31(3):251–269, 1997.

Munro, J. I., Papadakis, T., and Sedgewick, R. Deterministic Skip Lists. In Frederickson, G. N. (ed.), *Proc. SODA*, pp. 367–375. ACM/SIAM, 1992.

Pugh, W. W. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Commun. ACM*, 33(6):668–676, 1990.

Rubini, A. and Corbet, J. *Linux device drivers - covers Linux 2.4 (2. ed.).* O'Reilly, 2001.

Seidel, R. and Aragon, C. R. Randomized Search Trees. *Algorithmica*, 16(4-5):464–497, 1996.

Sleator, D. D. and Tarjan, R. E. Self-Adjusting Binary Search Trees. *J. ACM*, 32(3):652–686, 1985.

Sun, B., Lee, R., Hajiesmaili, M., Wierman, A., and Tsang, D. Pareto-optimal learning-augmented algorithms for online conversion problems. *Advances in Neural Information Processing Systems*, 34:10339–10350, 2021.

Zeynali, A., Sun, B., Hajiesmaili, M. H., and Wierman, A. Data-driven Competitive Algorithms for Online Knapsack and Set Cover. In *Proc. AAAI*, pp. 10833–10841. AAAI Press, 2021.

A. Proofs of Theoretical Result

A.1. Proof of Proposition 2.2

Proof. Consider defining the predictions $\hat{\boldsymbol{f}}$ in a way that all items are predicted to have a frequency that is almost $1/n$ with a small additive factor that ensures item i has rank i. More precisely, frequency of access to item i is $\hat{f}_i = 1/n + \epsilon_i$ (ϵ_i could be negative), where $\epsilon_i < \epsilon_j$ for any $i < j$, and $\sum_i \epsilon_i = 0$ and $|\epsilon_i| \ll 1/n$. Given that the key of each item equals its predicted rank, the learning augmented treap resembles a single path, and its height and robustness are linear to n. Even when predictions are accurate, treap's total cost for accesses to item n, which is $m\hat{f}_n$, is at least m/n, and the total cost for item i is $m\hat{f}_i = m \cdot i(1/n + \epsilon_i)$. The amortized cost for a single request is thus lower bounded by $\sum_{i=1}^{n} i\,(1/n + \epsilon_i) = \Omega(n)$. The cost of the treap for m access operations is thus $\Omega(mn)$, while the entropy of $\hat{\boldsymbol{f}}$ is $O(\log n)$. We can conclude that the consistency of the learned treap is $\Omega(n/\log n)$. □

A.2. Proof of Proposition 2.3

Proof. Optimal consistency is proven in [Theorem 4.8] (Cao et al., 2023). To provide a lower bound for robustness, consider a highly skewed distribution for the predicted frequency predictions such that $\log\log \hat{f}_i \geq \log\log \hat{f}_{i-1} + 1$ for any $i \in [n]$. As a result, the random component of the priority, which is in (0,1), does not make any difference in the rank of items in the resulting tree. That is, the item with key i will have rank i, and the learned treap will be highly unbalanced, resembling a linear list and therefore, its robustness is n. □

A.3. Proof of Lemma 3.1

Proof. Any arbitrary item from class c can replicate at least `class-base(c-1)` times only if the number of additional replications of item i, due to stochastic replications, exceeds a certain lower bound:

$$\mathcal{D}(c) - \mathcal{D}(c-1) = \frac{\log\theta}{\log p} 2^c \leq \mathcal{X}(p).$$

The probability of this event is less than $p^{\frac{\log(\theta)}{\log(p)}2^c}$. Moreover, the maximum number of items in class c is θ^{-2^c}, given that the predicted frequency of items in class c is at least θ^{2^c}. Consequently, the expected number of items in class c that may potentially replicate at least `class-base(c-1)` times, denoted as $\mathbb{E}[\text{Vio}_c]$, is bounded by:

$$\mathbb{E}[\text{Vio}_c] \leq p^{\log(\theta)2^c/\log(p)} \cdot \theta^{-2^c} \leq 1.$$

□

A.4. Proof of Lemma 3.2

Proof. We partition the linked lists in `RobustSL` into K *layers*, one layer for each class c as follows. The *layer* of class 0 is formed by the lists with depth at most $\mathcal{D}(0)$, and for any $c \in [1, K]$, the layer of class c is formed by lists at depth in the range $(\mathcal{D}(c-1), \mathcal{D}(c)]$. Since every item of class c is replicated in at least $\mathcal{D}(K) - \mathcal{D}(c)$ lists, it appears in layers of all classes $\geq c+1$.

To bound the number of comparisons for accessing an item i, we devise an upper bound for the number of comparisons at any layer $c \leq c_i$. Comparisons at layer c involve items that are replicated at $\mathcal{D}(c)$ but not at $\mathcal{D}(c-1)$. This is because items replicated at $\mathcal{D}(c-1)$ have been already examined in previous layers, and upon reaching layer c, the search domain is restricted to 2 consecutive items among them. Therefore, all comparisons at layer c involve items in class c as well as items from classes $\geq c+1$ that are replicated at $\mathcal{D}(c)$. By Lemma 3.1, the number of these latter items is at most 1, on expectation. To conclude, at layer c, we have at most $N_c + 1$ items with a replica at level $\mathcal{D}(c)$, each having further replicas following a geometric distribution with parameter p. In other words, they resemble a skip list, and searching among them takes at most $\frac{\log(N_c+1)}{p\log(1/p)} < \frac{2\log(N_c)}{p\log(1/p)}$, as in a regular skip list (Pugh, 1990).

Therefore, for the total search cost, we can write

$$\mathbb{E}[\texttt{Search}_{\texttt{RobustSL}}(i)] = \frac{2}{p\log(1/p)} \sum_{c\in[0,c_i]} \log(N_c) \tag{5}$$

$$= \frac{2}{p\log(1/p)} \sum_{[0,c_i]} (-2^c \log\theta) \tag{6}$$

$$< \frac{2\log\theta}{p\log p} 2^{c_i+1}. \tag{7}$$

When $c \leq K-1$, the right-hand side of the above inequality is at most $\frac{4}{p\log p}\log(\hat{f}_i)$, which follows directly from item classification.

When $c = K$, the total number of comparisons is the sum of comparisons in the layers above K and the number of comparisons at layer K. The first term is at most

$$\frac{2\log\theta}{p\log p}\cdot 2^K \leq \frac{2\log\theta}{p\log p}\cdot 2(\log n + \log\theta/\log p);$$

the first inequality follow from Equation (5), applying $c_i = K-1$, and the second inequality follows from the definition of K. Given that there are at most n items in class K, the number of comparisons at layer K is at most $\frac{\log n}{-p\log p}$ (as in a regular skip list). Therefore, the total number of comparisons is at most

$$\begin{aligned}
&\leq \frac{2\log\theta}{p\log p} 2(\log n + \log\theta/\log p) + \frac{\log n}{-p\log p} \\
&= \frac{4\log\theta - 1}{p\log p}(\log n) + \mathcal{O}(1).
\end{aligned}$$

□

A.5. Proof of Theorem 3.3

Proof. Suppose $\boldsymbol{f} = \hat{\boldsymbol{f}}$. According to Lemma 3.2, number of comparison made while searching an item i in class K is at most

$$\frac{4\log\theta - 1}{p\log p}(\log n) + \mathcal{O}(1) \leq \frac{4\log f_i}{p\log p};$$

the second inequality from Equation (1). In addition, again by Lemma 3.2, the number of comparisons made while searching an item i of class $c_i \leq K-1$ is bounded by $\frac{4\log f_i}{p\log p}$. Therefore, the total number of comparisons for the input sequence is at most $\sum_i f_i \cdot \frac{4\log f_i}{p\log p}$, which ensures a consistency of at most $\frac{-4}{p\log(p)}$. □

A.6. Proof of Theorem 3.4

Proof. According to Lemma 3.2 the maximum number of comparisons made while searching for any item is

$$\max_i \text{Search}_{\texttt{RobustSL}}(i) = \max\left(\frac{4}{p\log p}\log(\hat{f}_i), \frac{4\log\theta - 1}{p\log p}(\log n)\right) = \mathcal{O}(\log n).$$

This shows that `RobustSL` provides robustness of $\mathcal{O}(\log n)$, optimal robustness. □

A.7. Proof of Theorem 3.5

Proof. The expected number of comparisons made by `RobustSL` while searching for items with actual frequencies of $\boldsymbol{f}$ and predicted frequencies of $\hat{\boldsymbol{f}}$ is:

$$\sum_i f_i \cdot \texttt{Search}_{\texttt{RobustSL}}(i) = \sum_{\{i|c_i<K\}} f_i \cdot \texttt{Search}_{\texttt{RobustSL}}(i) + \sum_{\{i|c_i=K\}} f_i \cdot \texttt{Search}_{\texttt{RobustSL}}(i).$$

According to Lemma 3.2 we get:

$$\begin{aligned}\sum_i f_i \cdot \texttt{Search}_{\texttt{RobustSL}}(i) &= \sum_{\{i|c_i<K\}} f_i \cdot \texttt{Search}_{\texttt{RobustSL}}(i) + \sum_{\{i|c_i=K\}} f_i \cdot \texttt{Search}_{\texttt{RobustSL}}(i) \\ &\leq \sum_{\{i|c_i<K\}} f_i \cdot \frac{4}{p\log p}\log(\hat{f}_i) + \sum_{\{i|c_i=K\}} f_i \cdot (\frac{4\log\theta - 1}{p\log p}\log n) + \mathcal{O}(1) \\ &\leq \frac{-2\min(2\log\theta, \log p)}{p\log p} \cdot \texttt{Ent}(\boldsymbol{f}, \hat{\boldsymbol{f}}) + \mathcal{O}(1) = \mathcal{O}(\texttt{Ent}(\boldsymbol{f}, \hat{\boldsymbol{f}})).\end{aligned}$$

□

B. Additional Details of the Experiments

In this section, we provide additional details of the experimental setup.

Error models play a critical role in the performance of predicted-based data structures such as `RobustSL` or learning-augmented treaps. In (Lin et al., 2022), the authors define an error as the uncertainty in the frequency of item i. Specifically, the noisy predicted frequency of item i, $\hat{f}_i$, lies in the range $1/\Delta f_i \leq \hat{f}_i \leq \Delta f_i$, where Δ is a constant and representative of the error value. However, this error model suffers from the non-linear impact on the performance of predicted-based data structures. Precisely, for the mid-range error values, if $f_i \leq f_j$ denotes the frequency of item i and item j, then the expected rank of item i in the predicted ranks is also less than the expected rank of item j. As a result, even predictions with mid-range error values slightly lower the performance of data structures like learning-augmented treaps that consider the frequency rankings instead of individual frequency values of items.

To address this challenge, we consider an error model that swaps the ranks of high and low-ranked items for predictions with very high error values. Let δ_e be a metric that measures the accuracy of frequency predictions, where $\delta_e = 0$ indicates perfectly predicted frequencies and $\delta_e = 1$ denotes fully adversarial predicted frequencies. Using this error model, the predicted rank of item i is given by $\hat{r_i} = r_i \times (1 - \delta_e) + \delta_e \times (n - r_i + 1)$. In other words, the fully adversarial prediction model mirrors the rankings with respect to the median item. During our experiments, we use $\delta_e = 0.01$ as the high-accuracy prediction and $\delta_e = 0.9$ as the low-accuracy prediction.

Existing data structures often exhibit sensitivity to the order of queries, impacting the efficiency of search operations and other related tasks. For instance, splay trees demonstrate varying performance based on the order in which search queries are executed, while the structure of learned treaps is highly influenced by the frequency ordering of elements. It is our expectation that learned treaps will perform well in instances with perfect predictions due to their static optimality, while balanced BSTs will showcase robustness against fully adversarial inputs.

Consequently, we propose an error model that generates fully adversarial input sequences for prediction-based data structures like learned treaps and optimistic optimal binary search trees. These fully adversarial input sequences arise when high-frequency items are swapped with low-frequency items in the prediction result. By incorporating such challenging inputs into our experiments, we aim to assess the optimal robustness and consistency of `RobustSL` under diverse conditions.

Dynamic operations during experiments. Experiments on the synthetic dataset involve insert and delete queries. 80% of the items are randomly selected to be included in the initial data structure, and the ratio of insert/delete queries to search queries is 20%. Furthermore, insert and delete queries are randomly interleaved with search queries. The experiments are conducted using two methods: random order with perfect predictions ($\delta_e = 0$) and adversarial ranking with noisy predictions.

B.1. Result of experiments under static setting

In the static setting of Figure 4, the frequency ordering of elements is a crucial factor that can significantly impact the performance of data structures, such as learning-augmented treaps, which take it into account. When the items are ordered according to the adversarial frequencies, the performance of learning-augmented treap is severely affected, especially for a large number of unique keys, even though only 10% of tests included low-accuracy predictions. Results show that testing the dictionary of size $n = 2000$, the average number of comparisons of `RobustSL` is 7.1% more than of learning-augmented treaps under random frequency ordering while `RobustSL` achieves $45.1\times$ lower comparison per query when the predicted frequencies were adversarial. Finally, the result of testing the impact of the Zipfian parameter on the performance of data

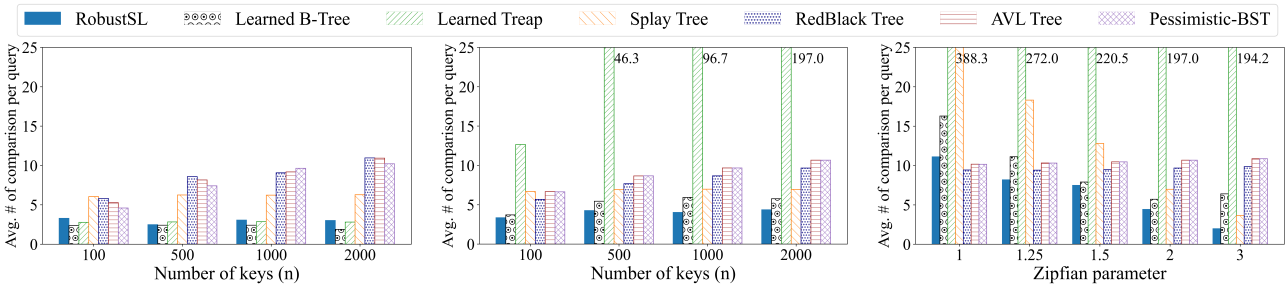

Figure 4. Average number of comparisons per query for `RobustSL` and baseline data structures for static evaluations under (left) random frequency ordering with perfect predictions, (center) adversarial frequency ordering with noisy predictions, and (right) adversarial frequency ordering with different value of Zipfian parameter. The frequency ordering of items impacts the performance of learning-augmented treaps while `RobustSL` shows its robustness against different conditions. To improve visibility, the y-axis range in the right figure is limited to 25, with actual values displayed next to bars exceeding this threshold for clarity.

structures in the static setting is consistent with the result of the same analysis under dynamic conditions.

B.2. Testing the impact of θ on performance of `RobustSL`

Figure 5 shows the impact of θ on the performance of `RobustSL` against the real-world dataset (BBC news article dataset). Similar to experiments conducted in Section 4.1, 0.368 was selected for parameter p, 40% of the dataset used as a training dataset, and the size of the adversarial dataset was 25% of the size of the training dataset. The results show that the value of parameter θ cannot significantly affect the performance of `RobustSL`.

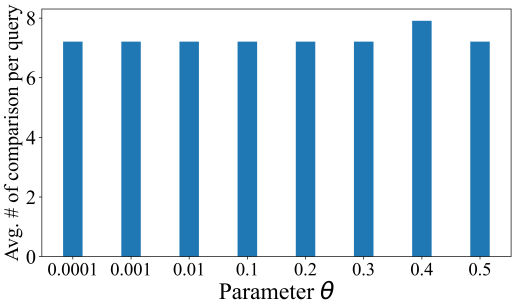

Figure 5. Average number of comparisons per query applied by `RobustSL` as a function of θ against the real-world dataset. θ shows negligible impact on the performance of `RobustSL`.