# A Tailored Regression for Learned Indexes: Logarithmic Error Regression

Martin Eppert
eppert@in.tum.de
Technische Universität München

Philipp Fent
fent@in.tum.de
Technische Universität München

Thomas Neumann
neumann@in.tum.de
Technische Universität München

## ABSTRACT

Although linear regressions are essential for learned index structures, most implementations use Simple Linear Regression, which optimizes the squared error. Since learned indexes use exponential search, regressions that optimize the logarithmic error are much better tailored for the use-case. By using this fitting optimization target, we can significantly improve learned index's lookup performance with no architectural changes.

While the log-error is harder to optimize, our novel algorithms and optimization heuristics can bring a practical performance improvement of the lookup latency. Even in cases where fast build times are paramount, log-error regressions still provide a robust fallback for degenerated leaf models. The resulting regressions are much better suited for learned indexes, and speed up lookups on data sets with outliers by over a factor of 2.

**ACM Reference Format:**
Martin Eppert, Philipp Fent, and Thomas Neumann. 2021. A Tailored Regression for Learned Indexes: Logarithmic Error Regression. In *Fourth Workshop in Exploiting AI Techniques for Data Management (aiDM'21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3464509.3464891

## 1 INTRODUCTION

Learned indexes are based around the idea of fitting the data distribution function to locate keys. Ideally, one would find the exactly matching function, which allows precisely calculating the position in the underlying data store. However, exactly matching functions are memory and compute intensive, which makes them unsuitable as an index. Instead, we trade precision for speed with an inexact function which gives an approximate position and use a refining search to find the data. The most popular function, used by many learned indexes [2, 4, 10, 11], is a linear model that is trained with Simple Linear Regression.

This standard technique in current learned indexes uses linear models that minimize the squared error. The popularity of the squared error stems primarily from widely known analytical solutions to minimizing it, which makes it simple to calculate, even tough its nominal value only has as weak connection to the optimization goal [5, 15]. In the case of learned indexes, we use exponential and binary search to find the exact data position. Consequently, when we intend to minimize the runtime, we need to consider the

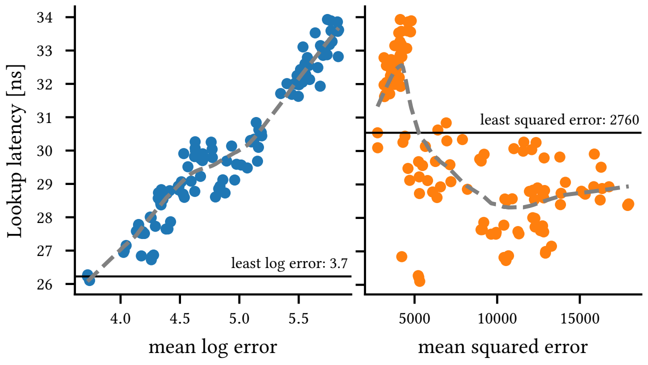

**Figure 1: Comparison of logarithmic and squared error regressions on normal distributed data. Minimizing the log instead of the squared error allows ~15 % lower lookup latency.**

logarithmic error: $\log_2(\epsilon)$. Since the current approaches optimize a different error, that does not directly capture the application, the resulting models have a brittle impedance mismatch between the quadratic model and the logarithmic runtime.

In our paper, we investigate the impact of this disconnect in the conventional metric and propose to use a logarithmic error regression instead. Figure 1 shows an experiment on normal distributed data to determine the correlation between lookup latency and the type of error. In this experiment, the lookup latency clearly follows the log-error, but the mean squared error does not correlate at all. Thus, we argue that the current practice optimizes a sub optimal metric, which results in brittle models that are susceptible to outliers. Least squares will worsen the model by accepting many small errors to better fit singular outliers. For example, consider one point with error 64, which is already a huge squared error, but its $\log_2$ error is easily overcome by a good fit for the remaining data.

When we study the behavior of the last-level refinement algorithms further, this disconnect becomes even more visible. Figure 2 shows the runtime of linear and exponential search, which most indexes use [9, 17, 21], with an increasing amount of prediction error. Unsurprisingly, we can observe that exponential search is much more robust to big errors, and is faster than linear search for errors exceeding five. In contrast, the squared error grows rapidly and quickly loses any connection even to the linear search.

This aspect of least squares regression is the reason for its susceptibility to outliers. Already adding a single point to the data set can significantly influence the outcome of the regression, which makes adversarial attacks trivial. Even without bad intentions, least squares regression is often sub optimal, especially with skewed

ACM ISBN 978-1-4503-8535-0/21/06.
https://doi.org/10.1145/3464509.3464891

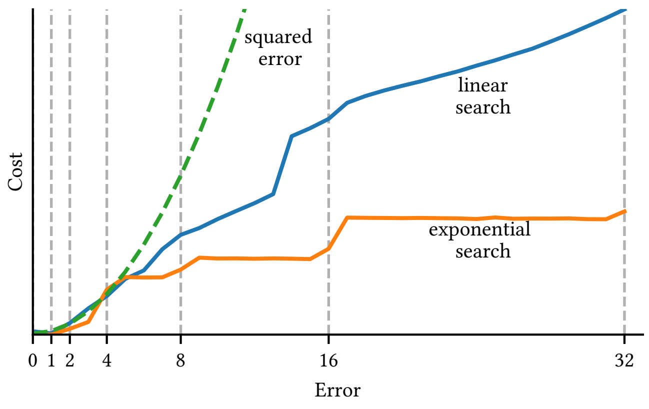

**Figure 2: Refinement time to find an element, depending on the prediction error. The squared error diverges quickly from exponential search time.**

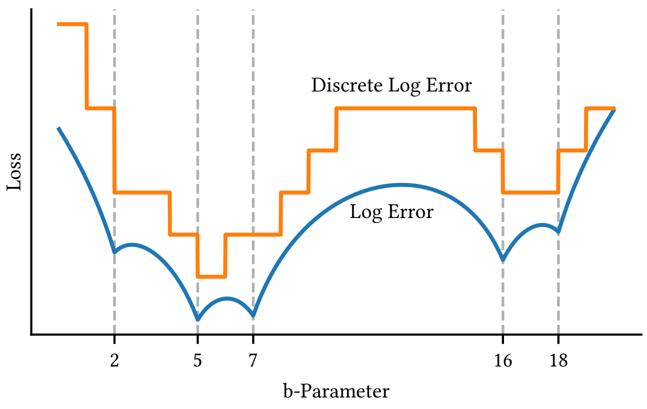

**Figure 3: Cross section of the linear regression loss landscape over the data set {2, 5, 7, 16, 18}. We fix the first parameter and consider all "horizontal" regressions $y = 0\,x + b$.**

data. When we instead optimize the logarithmic error, we get consistently better regressions, which are robust to outliers without filtering them out.

## 2 MINIMIZING THE LOGARITHMIC ERROR

Since the logarithmic error looks like the obvious optimization target, why do all existing implementations minimize the squared error instead? The main problem of the log-error is its non-convex loss function, thus optimizing it is computationally expensive. We define the loss of the continuous log-error function formally as:

$$L_{\log}(\mathrm{y}, \hat{\mathrm{y}}) \coloneqq \sum_{i=0}^{N} \log_2(1 + |y_i - \hat{y}_i|)$$

Respectively with a discrete error:

$$L_{\mathrm{dlog}}(\mathrm{y}, \hat{\mathrm{y}}) \coloneqq \sum_{i=0}^{N} \lceil \log(1 + |y_i - \hat{y}_i|) \rceil$$

A traditional approach to optimize such problems would be a gradient descent that minimizes the loss function, which we can calculate using the derivative of our loss function:

$$\frac{\partial L_{\log}(\mathrm{y}, \hat{\mathrm{y}})}{\partial \hat{\mathrm{y}}} = \sum_{i=0}^{N} \frac{1}{1 + |y_i - \hat{y}_i|} \cdot \mathrm{sign}(y_i - \hat{y}_i)$$

However, since our problem space in not convex, a gradient descent does not necessarily converge to the global optimum. In Figure 3, we illustrate this problem with the loss landscape of the logarithmic errors. We can observe that the plotted loss function has more than one minimum, and thus gradient descent will get stuck in one such local minimum. In other words, the loss is the product of the individual errors, which cannot be optimized with regular methods.

$$L_{\log}(\mathrm{y}, \hat{\mathrm{y}}) \coloneqq \log_2\left(\prod_{i=0}^{N}(1 + |y_i - \hat{y}_i|)\right)$$

Nevertheless, the error landscape contains an additional insight: All local minima coincide with data points. This means that we can vastly reduce the search space to minimize the logarithmic error, since we only need to consider linear regressions through two data points. In the following, we make use of this property to efficiently create logarithmic error regressions. The proof for this property can be lifted form the proof of the same property of the least absolute deviation (LAD) regression [16].

### 2.1 Optimal Solutions

**Naïve Algorithm** Following the insight that a regression crossing a point is a local minimum, we sketch a first naïve algorithm that brute-forces the optimal logarithmic regression: For a set of points $X$, we test all $X \times X$ combinations that define a line, and choose the regression with the smallest log-error. This method is well-known to minimize the least absolute deviation error [16].

For the asymptotic runtime of this algorithm, we need to consider the $O(n^2)$ combinations of candidate points. For each of the candidate regressions, we check the logarithmic error over the whole data set, which yields a total complexity of $O(n^3)$. For a practical application in an index structure, which have linear subsections of hundreds or thousands of points, this runtime is rather infeasible. Still, this is a significant improvement compared to NP-hard general optimization.

**Fast Discrete Logarithmic Error Regression** As an improvement over the naïve algorithm, we can exploit the fact that we are only interested in the discrete logarithmic error. The phenomenon we use here is that each point in the data can only produce $O(\log(\epsilon))$ distinct errors. Using this fact, we propose Algorithm 1 that efficiently enumerates the possible regressions in a circular sweep around a pivot point and discards any regressions with larger errors. Figure 4 visualizes this concept by pivoting regressions around the point (6,4). As a result, we can substitute the inner loop of the naïve algorithm by using a priority queue, which reduces a linear to a logarithmic factor, which improves the runtime to $O(n^2 \log(n) \log(\epsilon))$.

In practice, we round the values before we take the logarithm similar to how we have to round values to index into an array. This is problematic for the algorithm since the error is zero for $\epsilon \in [0, 1)$ and thus minima are not single points anymore. In this case, we can no longer assume that the optimal regression passes through two data points. However, it is sufficient to use the discrete logarithm without rounding since we are close to the true error.

**Algorithm 1:** Discrete log-error regression intersecting the pivot $p$.

```
Data: $K$ := data, sorted
Result: $a, b$ := optimal regression $y = a * x + b$ for the
        discrete logarithmic error
fun $dLogError(q, (a, b)) := \lceil \log(1 + |q_y - \lfloor a * q_x + b \rfloor|) \rceil$
$pq := MinPriorityQueue, e_{total} := 0$
for $k \in K$ do
    $e_{curr} = dLogError(k, (0, p_y))$
    $e_{total}$ += $e_{curr}$
    $slope$ := min $slope > 0$ of a line passing through $p$ with
      $dLogError(k, (slope, intercept)) \neq e_{curr}$
    $pq.insert(key = slope, value = (x, e_{curr}))$
end
$potential, e_{\min} := e_{total}$ ;      // Potential is an upper
 bound on the possible improvement in error
$a_{\min} := 0, b_{\min} := x_i.y$ $valid := false$
while $minCost \geq cost - potential \vee \neg valid$ do
    $slope_{prev}, (x, e_{curr}) := pq.pop()$
    $valid := slope_{prev} \neq pq.top()$
    $slope, intercept$ := line with min $slope > slope_{prev}$ of a
      line passing through $p$ with
      $dLogError(k, (slope, intercept)) \neq e_{curr}$
    update $e_{total}$ and $e_{curr}$ accordingly
    if $e_{total}$ is decreasing then
        $potential$ −= 1
    end
    if $valid \wedge e_{total} < e_{\min}$ then
        $e_{\min} := e_{total}$
        $a_{\min} := slope, b_{\min} := intercept$
    end
    $pq.insert(key = slope, value = (x, e_{curr}))$
end
return $(e_{\min}, a_{\min}, b_{\min})$
```

## 2.2 Approximate Solutions

Even though we already improve the runtime to find the optimal solution, finding it for large data sets is still expensive. Ideally, we need an algorithm with sub-quadratic runtime that still finds a near-optimal solution. In the following, we propose two faster, but approximate algorithms.

**Two Point Method** First, we propose a variant of the Two Point Method [16] which minimizes the logarithmic error. The regular Two Point Method minimizes the least absolute deviation by finding the optimal regression intersecting a randomly chosen data point. Since the optimal regression always intersects two points, we iterate this procedure on the second point, which yields a better regression with smaller or equal error. For least absolute deviation regressions, this method converges to the optimal solution, since its loss function is convex [12, 16, 18, 22].

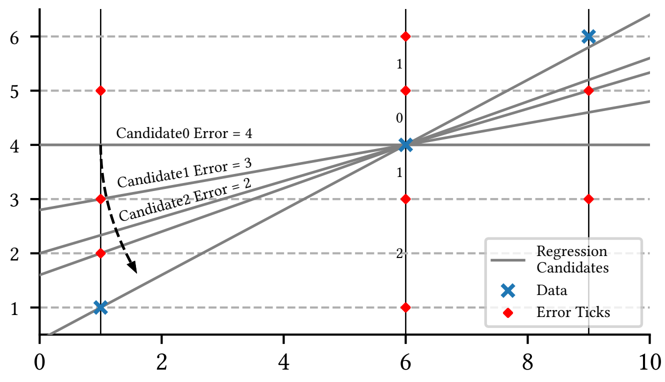

**Figure 4: Discrete log-error regression. We start with a horizontal line crossing the pivot, which we rotate until reaching a global minimum. Each rotational step takes $O(\log(n))$.**

To adapt the Two Point Method to log-errors, we can reuse our previous results to find optimal regressions intersecting a point. For continuous log-errors, this unfortunately needs $O(n^2)$, but for discrete errors, we can use our faster Algorithm 1. However, the loss-function is still non-convex, which implies that Two Point Method will not necessarily converge to the global optimum. Nevertheless, in our tests this approach consistently converges to near-optimal minima (cf. Table 1).

**Tournament Evaluation** With Algorithm 2 we devise an easy to use approximation to the log error regression problem called Tournament Evaluation, which uses a heuristic to determine the best regression from a set of candidates. The algorithm first picks $O(n)$ random regression candidates, which then compete in a single-elimination style tournament to determine the best logarithmic error regression. The space of all regressions is severely limited because the optimal regression must pass through a pair of data points.

For each round, we pair two neighboring candidates and eliminate the worse one. To accelerate the first rounds, we start with a small subset of points, on which we evaluate the regressions, but exponentially increase the number of points in later rounds. The reasoning behind this is that we can quickly eliminate very bad regressions and spend more time finding the best of the best. Thus, we find a heuristically good regression in $O(n \log(n))$.

Unfortunately, it is not possible to give guarantees on the quality of the resulting linear regression. However, in our experiments Tournament Evaluation performs well and consistently outperforms the Simple Linear Regression. Table 1 shows a qualitative analysis of the resulting regressions, which shows that the algorithms can keep the error within 1.5 % of the optimum.

**Refined Error Functions** All in all, Tournament Evaluation provides an efficient way to find good log-error regressions. However, using the pure log-error still has some limitations, e.g., at the bounds of the indexed underlying data. Since the regression prediction needs clamping to these bounds anyway, we can also incorporate this in the error function. For Tournament Evaluation this is a trivial

**Algorithm 2:** Tournament Evaluation to approximate the optimal linear regression with log-error.

```
Data: $X$ := data
Result: $a, b$ := approximate linear regression minimizing
        the logarithmic error with $y = a * x + b$
Function LogRegression($X$, $height$):
    // Generate the initial $O(n)$ random candidate set
    if $height == 1$ then
        $a, b := a \in X, b \in X, a \neq b$, chosen randomly
        return $a, b$
    end
    // Recurse to find competitors for the current round
    $a_1, b_1$ := LogRegression($X$, $height - 1$)
    $a_2, b_2$ := LogRegression($X$, $height - 1$)
    // Evaluate them on an increasingly bigger subset
    $e_1$ := sampleError($X$, $2^{height}$, $a_1$, $b_1$)
    $e_2$ := sampleError($X$, $2^{height}$, $a_2$, $b_2$)
    if $e_1 < e_2$ then
        return $a_1, b_1$
    else
        return $a_2, b_2$
    end
```

**Table 1: Qualitative analysis of the log-error fit. Each measurement shows the discrete log-error error compared to the optimum on a data set of 2000 keys. Evaluated methods: Simple Linear Regression (SLR), Least Absolute Deviation (LAD), Two Point Method (2P), Tournament Evaluation (TE).**

| Method | Facebook | Wiki | Normal | Uniform | Outlier |
|---|---|---|---|---|---|
| SLR[1] | 7.99 % | 7.34 % | 28.32 % | 4.66 % | 185.39 % |
| LAD | 2.78 % | 4.13 % | 7.55 % | 1.49 % | 3.14 % |
| 2P | 0.43 % | 0.81 % | 0.76 % | 1.19 % | 0.38 % |
| TE | 0.55 % | 0.60 % | 0.76 % | 0.93 % | 0.81 % |

change, which results in a slight performance advantage on small arrays.

## 3 TRADITIONAL METHODS

As sketched in Section 1, most previously proposed learned indexes use linear regressions that minimize the $L^2$-norm, i.e., the mean squared error (MSE). Some other methods are also used in learned indexes, such as interpolation, which share most pitfalls with the mean squared error regression [4, 10]. While we do not see a direct connection of this error to the expected performance of exponential search, optimizing the MSE can potentially bring the error close to the convergence point of $\epsilon = 0$. In the following, we first discuss the traditionally used algorithms for linear regression, before we compare them to our proposed approaches in Section 5.

**Simple Linear Regression** When looking at the publicly available implementations of learned indexes, the most frequently used implementation is Simple Linear Regression [2, 11]. Indeed, it is quite elegant and fits the optimal MSE regression in $O(n)$. Simple Linear Regression finds the optimal $\hat{a}, \hat{b}$ for data produced from a process described by $y_i = ax_i + b + \epsilon_i$. $\hat{a}, \hat{b}$ can be computed by the following equations, where $\bar{x}$ denotes the average value of $x$.

$$\hat{a} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \hat{b} = \bar{y} - (\hat{a}\bar{x})$$

However, Simple Linear Regressions does not come without problems: While it does find the optimal regression, this optimum is susceptible towards outliers. Since the errors are squared, even a small amount of data points can significantly disturb the optimal regression.

**Least Absolute Deviation** Since the squared error amplifies the errors, one obvious solution is to create a more robust regression using a different norm. For example, one could instead optimize the $L^1$-norm, which is the least absolute deviation:

$$L_{LAD}(\mathrm{y}, \hat{\mathrm{y}}) = \Sigma_i |y_i - \hat{y}_i|$$

Techniques optimizing it are also well-known, albeit already slower and usually require iteration. The two most common methods to optimize the LAD error are: Iteratively re-weighted least squares [1], and the Direct Descent method [22]. Still, the least absolute deviation weights outliers much higher than their performance penalty for exponential search.

**Robust Regression** In contrast, regressions over the logarithmic error are by definition much more robust concerning outliers. Since robust regressions are desirable for all kinds of applications, there also exist textbook algorithms that can guarantee robustness. A common robust regression method is the Theil-Sen estimator [20], which is available in many statistics libraries. Its desirable property is a guarantee of robustness, i.e., its breakdown point, which means it can handle about 30 % disturbed points, without degrading.

However, as we will see in the evaluation, the Theil-Sen regression overall performs slightly worse than MSE in regular workloads. Nevertheless, robustness is still advantageous to defend against edge cases where a single outlier can degrade the performance by over 2×. This problem was already highlighted in Kraska et al.'s foundational paper on learned indexes [11], where they propose a threshold fallback to regular B-Trees. This is often caused by a badly fitting MSE model, and could be avoided with robust regression.

## 4 PERFORMANCE TUNING

Compared to Simple Linear Regression, log-error regressions take more time to fit. Since indexes are used read-mostly, spending additional time to get a better fit is usually a good trade off. Nevertheless, it is still valuable to efficiently build the regression, and we identified two factors that benefit the log-regressions.

**Error Evaluation** An important primitive to calculate the log-error is a fast calculation of the logarithmic error. Especially for the discrete logarithm base two, this calculation can be very efficient. On integer distances, we use dedicated bit scan instructions that are present, e.g., in x86 and ARM instruction sets, and efficiently calculate the discrete logarithm:

```
num_bit - clz(1 + abs(x1 - x2)) - 1
```

On floating-point distances, we can also directly use the IEEE floating-point exponent. Both methods result in few executed instructions to calculate the error.

[1]SLR produces the optimal MSE, which can differ dramatically from the log-error.

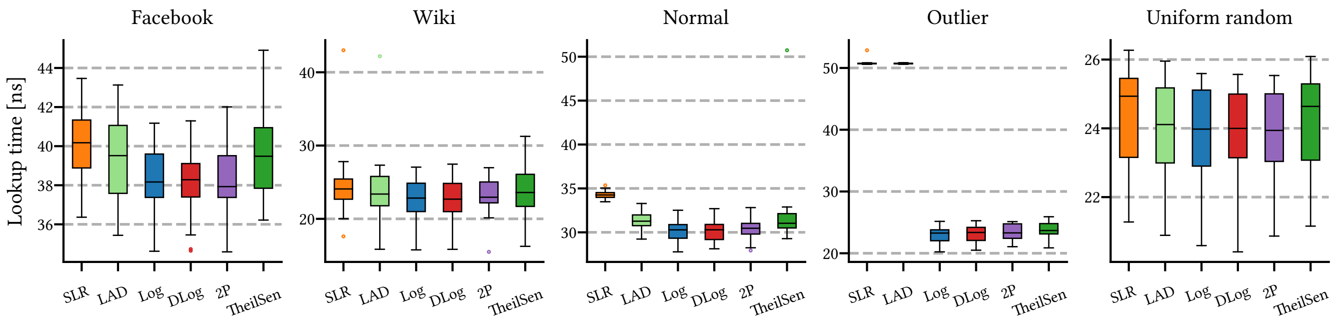

**Figure 5: Leaf Model Benchmark. Each box shows the lookup time of 1000 key subset linear regression constructed with: Simple Linear Regression (SLR), Least Absolute Deviation (LAD), log-error (Log) and discrete log-error (DLog) Tournament Evaluation, Two Point Method with discrete log-error (2P), and Theil Sen**

**Small Error Convergence** Another improvement to the build times is to run a tentative cheap regression, e.g., a Simple Linear Regression. When this already results in tiny errors, i.e., $0 \approx \epsilon \approx \epsilon^2 \approx \log_2(\epsilon)$, then any regression will converge to the same linear function, and we can finish with the cheaply calculated regression. This effect is additionally amplified by the behavior of the search functions with small errors that we could already observe in Figure 2. When the error is small enough, e.g., $\leq 4$, the search is dominated by the cache-line granular memory access that loads all searched elements simultaneously. Nevertheless, when the errors are larger, logarithmic error regression is strictly better than a MSE regression.

## 5 EVALUATION

In the following, we evaluate the effect of the discussed alternative regressions on the performance of learned indexes. First we run micro-benchmarks on isolated leaf models, which gives a clear view into the affected component of learned indexes. Afterwards, we integrate these methods in the open-source RMI [11] to measure the effect on the whole system.

All benchmarks were executed with a single thread on an Intel Xeon E5-2660v2 CPU and repeated 30 times to measure performance with warm caches. The tested data sets are largely from the Search on Sorted Data (SOSD) benchmark[2]. Our implementation of the recursive model indexes is based on the reference implementation[3], with some additional data sets and log-regression: https://github.com/umatin/LogarithmicErrorRegression

For our micro benchmarks, we segment the data set into leaf models of linear regressions over 1000 keys each. Then, we create a linear model for predictions using each of the presented methods: Simple Linear Regression (SLR), Tournament Evaluation optimizing the log-error (Log) and the discrete log-error (DLog), the Two Point Method optimizing the log-error (2P), and Theil-Sen regression. After using the model to get a starting position, we find the matching key using exponential search.

In addition to the SOSD data sets, we also use a generated outlier data set, which is intended to test the robustness of the regressions. Here, we generate a uniform random data distribution, and deliberately introduce a single point that differ strongly from the linearity assumption of the regressions. This single outlier is sufficient to inflate the squared error, which causes bad SLRs. While this occurs in few leaf models of regular data sets, it showcases the potential improvements that are possible in the cases where it does.

For the micro benchmark, Figure 5 shows the resulting lookup times in leaf models trained with different linear regressions, and Table 2 breaks down the mean and median speedup over Simple Linear Regression. Over all data sets, Simple Linear Regression performs worst, with logarithmic error regression consistently improving the performance by a few percent. Even in the case of uniform random data, where even a Simple Linear Regression usually matches the data well, the leaves having outliers by chance can still be improved by log-error regression. We see the most pronounced speedup with the outlier distribution, where the robust regressions perform at 2.2×.

In this benchmark, we measured three variants that optimize the logarithmic error: Log, which optimized the continuous log-error, DLog that optimizes the discrete version calculated with `clz`, and 2P, which uses the Two Point Method. Overall, they perform roughly equal and all three find good regressions. However, their build times can differ quite dramatically.

Figure 6 shows the distribution of build times for each of the methods on example leafs with 1000 keys. Unsurprisingly, Simple Linear Regression has much shorter build times, and is over 50× faster than the other methods. The robust Theil-Sen regression even takes around 5 ms to build a single leaf model, even with a reduced subset of 10 k point combinations. Optimizing the logarithmic error using 3 steps of Two Point Method is somewhat quicker, but is still outperformed by the tournament evaluation variants. Especially with the optimizations using the discrete logarithm, building the leaf models works in under 0.1 ms, which makes log-error feasible in RMIs.

**Impact on RMI** The previous micro benchmarks have already shown that the linear regression can have a significant impact on the performance of individual leaf models. In the following, we look at the complete RMI on workloads that use a linear regression on their leaf models. For our experiments, we use the RMI configuration

[2]https://github.com/learnedsystems/SOSD
[3]https://github.com/learnedsystems/RMI

**Table 2: Leaf model speedups over Simple Linear Regression**

| | | Speedup | |
|---|---|---|---|
| Method | Data set | Mean | Median |
| LAD | Facebook | 1.5 % | 2.0 % |
| Log | | 4.8 % | 6.0 % |
| DLog | | 4.9 % | 5.1 % |
| 2P | | 4.5 % | 6.7 % |
| TheilSen | | 1.1 % | 1.0 % |
| LAD | Wiki | 2.2 % | 1.7 % |
| Log | | 7.9 % | 5.2 % |
| DLog | | 7.7 % | 5.5 % |
| 2P | | 6.2 % | 4.5 % |
| TheilSen | | 3.4 % | 0.9 % |
| LAD | Normal | 9.3 % | 9.5 % |
| Log | | 13.5 % | 13.0 % |
| DLog | | 13.2 % | 12.8 % |
| 2P | | 12.7 % | 12.5 % |
| TheilSen | | 6.6 % | 10.1 % |
| LAD | Outlier | 0.2 % | 0.0 % |
| Log | | 119.7 % | 117.1 % |
| DLog | | 118.7 % | 116.4 % |
| 2P | | 117.0 % | 117.3 % |
| TheilSen | | 113.8 % | 112.5 % |
| LAD | Uniform | 1.4 % | 3.1 % |
| Log | | 2.2 % | 3.9 % |
| DLog | | 2.0 % | 3.4 % |
| 2P | | 1.9 % | 3.8 % |
| TheilSen | | 0.7 % | 0.8 % |

of the SOSD benchmark [9], which already gives state-of-the-art performance on the evaluated workloads. The SOSD workloads each consist of 200 million data points, and we additionally generate a same sized synthetic workload using a gaussian mixture model with uniformly random distributed means. The gaussian mixture data set is a distribution that illustrates a bad case for the Simple Linear Regression. On the four workloads we present here, the first layer differs for each workload, but the second layer is always a linear regression.

Table 3 shows a comparison of Simple Linear Regression, which the reference implementation uses, and our improved log-regression that optimizes the discrete log-error with a Tournament Evaluation, which consistently improves the RMI performance. While the improvements of the leaf models of the Wiki data translate quite well, the speedup on the Facebook data set is less pronounced. While we expect a bit less speedup due to unrelated overhead in the first layer, we would expect a speedup that is in the same order of magnitude. In a preliminary investigation, we found that the data set contains large numbers spanning the whole 64-bit number space, which might be a factor that interferes with the precision of our linear models. Overall, the log-error regression results in a geomean speedup of 5.8 %.

**Table 3: RMI performance with a second layer of 200k linear regression leaf models.**

| | | Lookup [ns] | | | Build [s] | | |
|---|---|---|---|---|---|---|---|
| Data set | 1st Layer | SLR | Log | Speedup | SLR | Log | Slowdown |
| Facebook | linear | 245.7 | 244.1 | 0.7 % | 23 | 65 | 2.8× |
| Wiki | linear | 172.1 | 160.6 | 7.2 % | 24 | 67 | 2.8× |
| Osm | cubic | 399.4 | 383.4 | 4.2 % | 27 | 122 | 4.5× |
| Gaussian Mixture | linear | 296.7 | 266.0 | 11.5 % | 21 | 47 | 2.2× |

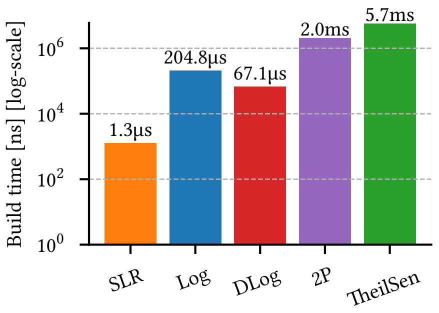

**Figure 6: Build times for 1000 key leaf models. The build times do not differ significantly on different data sets.**

However, the lookup performance improvements come with an increased training overhead. As Table 3 shows, the RMI built times increase roughly by a factor of 2–3× when using our proposed Tournament Evaluation. An outlier is the OSM data set which needs 4.5× the build time, which is caused by a relatively bad fitting first layer with low information gain. This causes comparatively large leaf models, which take more time to construct proportionally due to the $O(n \log(n))$ build time of the Tournament Evaluation. We argue that RMIs are already expensive to construct, and their primary use-case is read-heavy workloads, where our improvements are well worth the extra training time. Nevertheless, we are convinced that we can improve the training times by using log-error regressions only on leaf nodes that benefit significantly.

## 6 RELATED INDEX STRUCTURES

Even though learned index structures are in the spotlight of current database research [3, 6–8, 14, 19], we are not aware of any indexes that optimize the logarithmic error. Since our improvements are mainly targeted at the used linear regressions, we argue that they should translate to other related indexes besides the RMI.

Similar to the RMI, ALEX, the updatable adaptive learned index [2] uses Simple Linear Regression to build its leaf models. However, preliminary experiments on their open-source implementation resulted in no significant differences. The reason behind these results is their use of model-based insertions, which can already eliminate most of the linear regression error. ALEX uses gapped arrays to distribute extra space for insertions between elements, which also allows them to insert almost all already existing keys at their correctly predicted location. Then, the resulting model has zero error, where any previous improvements of the regression have little impact.

The single-pass build approach RadixSpline [10], in contrast, does not use linear regression. Instead, they build a spline interpolation with a maximum error bound, essentially bounding the $L^{\infty}$-norm. While this is an effective technique to find spline points, we argue that it does not optimize the right error. However, their focus on fast construction makes more expensive error calculations not viable.

In contrast, the Piecewise Geometric Model (PGM) index [21] uses linear models, and thus exponential search, at every level of the search tree. When using Simple Linear Regression on each level of this tree, the effects of poorly fitting models amplify. However, we did not investigate this further, since their construction algorithm for inner models differs significantly from our approach.

Ryan et al. [13] suggest creating linear models that factor in cache line access costs. Adapting the presented Two Point Method for log-errors should be seamlessly possible.

## 7 CONCLUSION

In our paper, we clearly demonstrated that Simple Linear Regressions are sub optimal for learned indexes. In contrast to minimizing the squared error, a minimum log-error optimizes the relevant error for exponential search, which we consider optimal. Compared to Simple Linear Regression, our approach speeds up average lookup performance by around 6 % and improves extreme cases by 2.2×.

While the overhead in build times makes this approach not suitable in all cases, this is less a concern for read-heavy workloads. Nevertheless, we are convinced that a log-regression can also be very beneficial for the edge cases where squared and log-error differ widely. With a fallback log-error regression, learned indexes are significantly more robust and additionally get a performance improvements whenever their time budget allows more optimization.

## ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 725286).

## REFERENCES

[1] C. Sidney Burrus. Dec 17, 2012. Iterative Reweighted Least Squares. *OpenStax CNX.* (Dec 17, 2012). http://cnx.org/contents/92b90377-2b34-49e4-b26f-7fe572db78a1@12.
[2] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David B. Lomet, and Tim Kraska. 2020. ALEX: An Updatable Adaptive Learned Index. In *SIGMOD Conference.* ACM, 969–984.
[3] Jens Dittrich, Joris Nix, and Christian Schön. 2020. There is No Such Thing as an "Index"! or: The next 500 Indexing Papers. *CoRR* abs/2009.10669 (2020).
[4] Paolo Ferragina and Giorgio Vinciguerra. 2020. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. *Proc. VLDB Endow.* 13, 8 (2020), 1162–1175.
[5] Minos N. Garofalakis and Amit Kumar. 2005. Wavelet synopses for general error metrics. *ACM Trans. Database Syst.* 30, 4 (2005), 888–928.
[6] Ali Hadian and Thomas Heinis. 2020. MADEX: Learning-augmented Algorithmic Index Structures. In *AIDB@VLDB.*
[7] Ali Hadian, Ankit Kumar, and Thomas Heinis. 2020. Hands-off Model Integration in Spatial Index Structures. In *AIDB@VLDB.*
[8] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *Proc. VLDB Endow.* 13, 7 (2020), 992–1005.
[9] Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. 2019. SOSD: A Benchmark for Learned Indexes. *NeurIPS Workshop on Machine Learning for Systems* (2019).
[10] Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. 2020. RadixSpline: a single-pass learned index. In *aiDM@SIGMOD.* ACM, 5:1–5:5.
[11] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *SIGMOD Conference.* ACM, 489–504.
[12] Yinbo Li and Gonzalo R. Arce. 2004. A Maximum Likelihood Approach to Least Absolute Deviation Regression. *EURASIP J. Adv. Signal Process.* 2004, 12 (2004), 1762–1769.
[13] Ryan Marcus, Andreas Kipf, Alexander van Renen, Mihail Stoian, Sanchit Misra, Alfons Kemper, Thomas Neumann, and Tim Kraska. 2020. Benchmarking Learned Indexes. *Proc. VLDB Endow.* 14, 1 (2020), 1–13.
[14] Ryan Marcus, Emily Zhang, and Tim Kraska. 2020. CDFShop: Exploring and Optimizing Learned Index Structures. In *SIGMOD Conference.* ACM, 2789–2792.
[15] Thomas Neumann and Sebastian Michel. 2008. Smooth Interpolating Histograms with Error Guarantees. In *BNCOD (Lecture Notes in Computer Science, Vol. 5071).* Springer, 126–138.
[16] Kristian Sabo and Rudolf Scitovski. 2008. The best least absolute deviations line - Properties and two efficient methods for its derivation. *The ANZIAM Journal* 50 (10 2008), 185 – 198. https://doi.org/10.1017/S1446181108000345
[17] Peter Van Sandt, Yannis Chronis, and Jignesh M. Patel. 2019. Efficiently Searching In-Memory Sorted Arrays: Revenge of the Interpolation Search?. In *SIGMOD Conference.* ACM, 36–53.
[18] E. J. Schlossmacher. 1973. An Iterative Technique for Absolute Deviations Curve Fitting. *J. Amer. Statist. Assoc.* 68, 344 (1973), 857–859.
[19] Naufal Fikri Setiawan, Benjamin I. P. Rubinstein, and Renata Borovica-Gajic. 2020. Function Interpolation for Learned Index Structures. In *ADC (Lecture Notes in Computer Science, Vol. 12008).* Springer, 68–80.
[20] Henri Theil. 1950. A rank-invariant method of linear and polynomial regression analysis. *Indagationes Mathematicae* 12, 85 (1950), 173.
[21] Giorgio Vinciguerra, Paolo Ferragina, and Michele Miccinesi. 2019. Superseding traditional indexes by orchestrating learning and geometry. *CoRR* abs/1903.00507 (2019).
[22] G. O. Wesolowsky. 1981. A new descent algorithm for the least absolute value regression problem. *Communications in Statistics - Simulation and Computation* 10, 5 (1981), 479–491.