LINEAR-TIME SEGMENTED REGRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

We present a linear-time algorithm for min-ϵ segmented regression, which fits a piecewise-linear function with a prescribed number of segments k to a given dataset while minimizing the residual sum of squares. Such learning settings arise in many applications, including the general compression of time series data, the detection of economic indicators, or the identification of abrupt shifts in climate data. Exact solutions can be obtained via dynamic programming. However, the runtime scales quadratically with the dataset size. Recently, a greedy approach based on merging and subsequently fine-tuning segments has been proposed, which achieves a log-linear runtime in the number of data points. In this work, we propose an approach based on a greedy splitting and fine-tuning of segments, which exhibits a linear runtime with respect to the dataset size. In our experiments, we assess the runtime behavior and the achieved quality of our scheme and compare the results with dynamic programming and the recently proposed greedy approach, using both synthetic and real-world data. Our results indicate that our method scales linearly in practice, runs faster than the competing heuristic, and matches the exact solutions more often than the other approximation scheme.

1 INTRODUCTION

Segmented regression models the relationship between independent input and dependent output variables using a piecewise function, with breakpoints regarding the independent input variables. More precisely, the goal of *min-ϵ segmented regression* is to fit a piecewise function with a predefined number $k \in \mathbb{N}$ of segments. A common approach to address this problem is to use piecewise-linear functions and to minimize the mean-squared error (MSE) between model predictions and data points. For one segment (i.e., $k = 1$), this reduces to classical ordinary least squares (OLS) regression, which can be solved in $\mathcal{O}(Nd^2)$ time for N data points in $\mathbb{R}^d$. For $k \geq 2$, finding the best breakpoints adds complexity. A naïve approach is based on assessing the model fit for all $\binom{N-1}{k-1}$ possible breakpoint combinations, resulting in a polynomial runtime behavior, quickly becoming infeasible for large N or k. Segmented regression has roots in many disciplines, including image processing, where it is used for polygonal approximations of planar curves (Ramer, 1972), or cartography, where it is used to simplify digitized lines (such as coastlines) by resorting to piecewise fits to reduce the number of data points (Douglas & Peucker, 1973). Over the years, applications have extended to various other domains, such as ecological science (Shao & Campbell, 2002), veterinary science (Hawkins et al., 2026), econometrics (Yamamoto & Perron, 2013), grid computing (Wies et al., 2026), data mining (Keogh et al., 2001), or climatology (Tomé & Miranda, 2004).

More than two decades ago, a dynamic programming approach (Bai & Perron, 2003) was proposed, which can be used to identify an exact solution in quadratic time (Acharya et al., 2016b). However, as data volumes continue to grow, a quadratic runtime behavior can still become infeasible in certain application areas, especially in case many such models have to be fitted, such as remote sensing, where billions of such models may need to be fitted (DeVries et al., 2015; de Jong et al., 2013; Gieseke et al., 2020; Hislop et al., 2019). As a consequence, the development of fast approximation schemes is still the subject of ongoing research. Recently, Lößer et al. (2025) proposed such a scheme that is based on greedily merging segments, along with a subsequent fine-tuning step of the identified breakpoints. The approach yields very competitive results, while exhibiting a sub-quadratic runtime of $\mathcal{O}(N \log(N))$ for N data points.

In this work, we propose an approximation scheme for min-ϵ segmented regression that exhibits a linear runtime w.r.t. the number N of data points. Essentially, rather than greedily merging segments

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

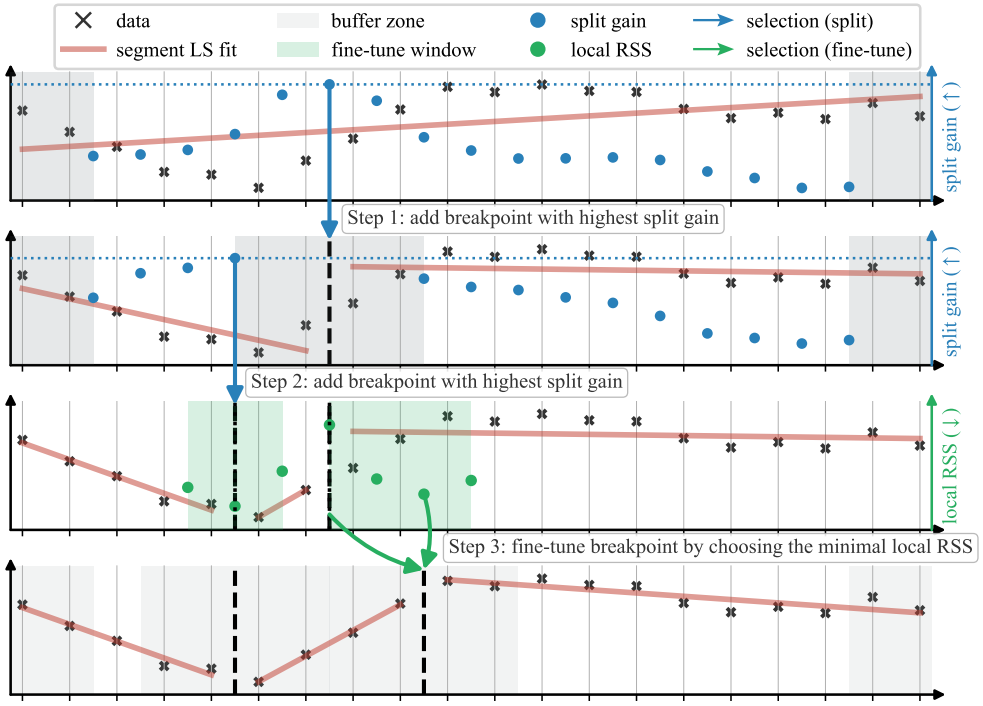

Figure 1: Illustration of our approach on synthetic data (black crosses) with $k = 3$ segments. The algorithm alternates between two procedures. *(i) Split:* the position with the highest split gain (blue, right axis) becomes a new breakpoint (black dashed). *(ii) Fine-tune:* each interior breakpoint is moved within a local window (green) to minimize the local RSS (green, right axis). Grey bands mark the buffer zones that guarantee a minimum number of points per segment. The first breakpoint requires no fine-tuning (see detailed description).

as done by Lößer et al. (2025), our approach greedily splits segments. The overall workflow is sketched in Figure 1. The dataset is initially treated as a single segment, and the first split point is chosen in a way that maximizes the gain. The method then alternates between inserting additional breakpoints, again guided by the splitting gain, and refining the positions of all existing breakpoints, following Lößer et al. (2025). The procedure terminates after a fine-tuning step, once the data have been divided into k segments. Our experiments show that this simple yet effective scheme closely matches the exact dynamic programming baseline, recovering the optimal solution more often than its direct competitor while exhibiting a runtime of $\mathcal{O}(N)$ for a fixed number k of segments.[1]

2 Background

We first introduce the problem setup of segmented regression and then discuss related work.

2.1 Segmented Regression

Segmented regression is mainly studied in two variants, min-ϵ and min-$\#$ segmented regression; in this work we focus on the former.

2.1.1 Notation

Piecewise-linear Functions. Let $d \in \mathbb{N}$ be the input dimension and $j \in \{1, \dots, d\}$ a coordinate. A function $f : \mathbb{R}^d \to \mathbb{R}$ is called *piecewise-linear* with respect to j with at most $k \in \mathbb{N}$ segments, if there is a partition of the real line into k pairwise disjoint intervals $\mathbb{I}_1, \dots, \mathbb{I}_k \subset \mathbb{R}$ such that for

[1]The runtime also scales linearly with k, effectively replacing the competitor's $\log N$ factor with $k \cdot d^2$. For moderate k and small d, this yields substantial practical speed-ups, as demonstrated by our experiments.

every $i \in \{1, \dots, k\}$, there exists a $\boldsymbol{w}_i \in \mathbb{R}^d$ with $f(\boldsymbol{x}) = \boldsymbol{w}_i^{\mathrm{T}} \cdot \boldsymbol{x}$ for all $\boldsymbol{x} \in \mathbb{R}^d$ with $x_j \in \mathbb{I}_i$.

$$\mathbb{L}_{k,j}^d \coloneqq \{f : \mathbb{R}^d \to \mathbb{R} \mid f \text{ is piecewise-linear w.r.t. coordinate } j \text{ with at most } k \text{ segments}\} \quad (1)$$

denotes the corresponding function space. Note that for $k = 1$, the only admissible partition is $\mathbb{I}_1 = \mathbb{R}$, so that $f(\boldsymbol{x}) = \boldsymbol{w}_1^{\mathrm{T}} \cdot \boldsymbol{x}$ for all $\boldsymbol{x} \in \mathbb{R}^d$, i.e. f is linear. Consequently, $\mathbb{L}_{1,j}^d$ coincides with the set of linear functions for every j, and we denote it simply by $\mathbb{L}^d$.

Example I. For $d = 1$, $f : \mathbb{R} \to \mathbb{R}$ is piecewise-linear, iff there are a partition $\mathbb{R} = \cup_{i=1}^k \mathbb{I}_i$ into pairwise disjoint intervals and slopes $w_i \in \mathbb{R}$ with $f(x) = w_i \cdot x$ for $x \in \mathbb{I}_i$.
Example II. Note that the above example does not include intercepts. Adding intercepts increases the input dimension to $d = 2$, as in plain linear regression. More precisely, for a piecewise-linear function $f : \mathbb{R}^2 \to \mathbb{R}$ with respect to the coordinate $j = 2$, there exist a partition as in Example I and intercepts $w_0^{(i)} \in \mathbb{R}$ and slopes $w_1^{(i)} \in \mathbb{R}$ such that $f((1, x)^{\mathrm{T}}) = w_0^{(i)} + w_1^{(i)} \cdot x$ for $x \in \mathbb{I}_i$.
Example III. More generally, as for plain multiple linear regression, piecewise-polynomial functions of degree $d - 1$ arise from piecewise-linear functions $f : \mathbb{R}^d \to \mathbb{R}$ with respect to the coordinate $j = 2$, via the polynomial encoding $\mathrm{poly}_{\text{d-1}} : \mathbb{R} \to \mathbb{R}^d$, $x \mapsto (1, x, \dots, x^{d-1})$. Appendix A provides a detailed discussion.

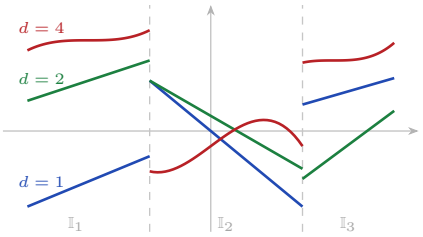

Figure 2: Examples of piecewise-linear functions with different input dimension d on a common partition into $k = 3$ intervals. For $d = 1$ every piece lies on a line through the origin, for $d = 2$ intercepts are added, and for $d = 4$ the pieces are cubic.

Figure 2 visualizes exemplary functions from Examples I–III.

Residual Sum of Squares. Let $((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)) \in (\mathbb{R}^d \times \mathbb{R})^N$ be a dataset of size N with data matrix $\boldsymbol{X} = (\boldsymbol{x}_1 \ \dots \ \boldsymbol{x}_N)^{\mathrm{T}} \in \mathbb{R}^{N \times d}$ and observation vector $\boldsymbol{y} = (y_1, \dots, y_N) \in \mathbb{R}^N$. The *residual sum of squares* (RSS) of a function $f : \mathbb{R}^d \to \mathbb{R}$ on this dataset is given by

$$\mathrm{RSS}(f, \boldsymbol{X}, \boldsymbol{y}) \coloneqq \sum_{n=1}^{N} \big(f(\boldsymbol{X}_{n,:}) - y_n\big)^2 \in [0, \infty), \quad (2)$$

where $\boldsymbol{X}_{n,:} \coloneqq \boldsymbol{x}_n \in \mathbb{R}^d$ denotes the n-th feature vector for an index $n \in \{1, \dots, N\}$. To simplify the notation, we henceforth specify dataset splits by breakpoint indices rather than specific coordinate values, following Acharya et al. (2016a) and Lößer et al. (2025). More precisely, we assume that there exists a splitting coordinate $j \in \{1, \dots, d\}$ along which the data are strictly increasing, i.e. $\boldsymbol{X}_{1,j} < \dots < \boldsymbol{X}_{N,j}$.[2] Since j is fixed throughout the remainder of this work, we omit it from the notation and write $\mathbb{L}_k^d$ for the set of piecewise-linear functions with at most k segments. Furthermore, for two indices $b \leq b' \in \{1, \dots, N\}$, we denote by $\boldsymbol{X}_{b:b'} \coloneqq (\boldsymbol{x}_b \ \dots \ \boldsymbol{x}_{b'-1})^{\mathrm{T}} \in \mathbb{R}^{(b'-b) \times d}$ and $\boldsymbol{y}_{b:b'} \coloneqq (y_b \ \dots \ y_{b'-1}) \in \mathbb{R}^{b'-b}$ the feature matrix and the target vector of the segment ranging from index b (included) to index b' (excluded), which contains $b' - b$ data points. The RSS of an optimal linear regression on this segment is given by

$$\mathrm{RSS}(b, b') \coloneqq \min_{f \in \mathbb{L}^d} \mathrm{RSS}(f, \boldsymbol{X}_{b:b'}, \boldsymbol{y}_{b:b'}) \in [0, \infty), \quad (3)$$

where we have dropped $\boldsymbol{X}$ and $\boldsymbol{y}$ from the definition of $\mathrm{RSS}(b, b')$ for the sake of readability. Note that, in general, we have $\mathrm{RSS}(b, b') = 0$ whenever the segment contains at most d data points, since a linear model with d parameters interpolates any d points in general position.

2.1.2 Problem Setup

In min-ϵ segmented regression, the number of segments $k \in \mathbb{N}$ is given and the task is to minimize the error over all piecewise-linear functions with at most k segments, i.e., one aims at finding

$$\hat{f}_{\min-\epsilon} \coloneqq \operatorname*{arg\,min}_{f \in \mathbb{L}_k^d} \mathrm{RSS}(f, \boldsymbol{X}, \boldsymbol{y}) \in \mathbb{L}_k^d. \quad (4)$$

[2] This assumption is often satisfied in practice, e.g. for time-series with only one data point per timestamp. All algorithms can be extended to the general case.

Note that, under the index-splitting assumption, this amounts to selecting $k-1$ breakpoints out of $N-1$ possible positions, since ordinary least squares then yields the optimal segment parameters in closed form. For a $\hat{f}_{\text{min}-\epsilon}$, the induced error is given by $\epsilon \coloneqq \text{RSS}(\hat{f}_{\text{min}-\epsilon}, \boldsymbol{X}, \boldsymbol{y}) \in [0, \infty)$.

2.2 Related Work

There are several approaches to segmented regression resulting in exact or heuristic solutions. An exact approach based on dynamic programming proposed by Bai & Perron (2003) runs in quadratic time w.r.t. the dataset size N. There exists work to reduce its runtime, e.g. by using a pruned version, which leads to an expected linear runtime under certain conditions, but the worst case runtime remains quadratic (Rigaill, 2015; Killick et al., 2012).

As a quadratic runtime is infeasible for large datasets, faster heuristic algorithms have been developed, which can be divided into two groups: bottom-up and top-down approaches. Top-down algorithms start with one segment and iteratively add segments, e.g., by greedily splitting segments with the best split gain (Bai, 1997). For piecewise constant functions, there are approaches to greedily split on random local windows to not overlook small segments (Fryzlewicz, 2014). Bottom-up algorithms, by contrast, begin with the maximum possible number of segments and iteratively reduce them by merging. A greedy variant, which at each step merges the pair of segments incurring the lowest merging cost, was, e.g., proposed by Keogh (1997). Variants of this approach merge multiple segments at once and provide theoretical guarantees for the convergence rate (Acharya et al., 2016a). The state-of-the-art algorithm in min-ϵ segmented regression is based on efficient greedy merging and a final local fine-tuning step (Lößer et al., 2025).

3 Approach

We present an algorithm that is a combination of existing algorithms: it is a top-down greedy algorithm that resorts to the local fine-tuning from Lößer et al. (2025) after every splitting step. Below, we first review the heuristic algorithm of Lößer et al. (2025) before outlining our modifications. We refer to Appendix B for a brief description of the Dynamic Programming (DP) approach proposed by Bai & Perron (2003), which is used as one of the baselines in our experimental evaluation.

3.1 Greedy Merging and Fine-Tune (Merge+FT)

We denote the Greedy Merging and Fine-Tune algorithm (Lößer et al., 2025) by Merge+FT. It consists of two steps: greedy bottom-up merging followed by fine-tuning of the breakpoints.

Greedy Merging. Starting from an initial segmentation into segments of size d, the two consecutive segments with smallest merging cost are merged in each iteration, until k segments remain. For three consecutive breakpoints $b_{i-1} < b_i < b_{i+1}$, the merging cost of the two segments separated by b_i is the resulting increase in the total RSS,

$$\text{RSS}(b_{i-1}, b_{i+1}) - \big(\text{RSS}(b_{i-1}, b_i) + \text{RSS}(b_i, b_{i+1})\big) \;\geq\; 0, \tag{5}$$

so that merging the cheapest pair is the locally least harmful choice.

Fine-tuning. Greedy merging only removes breakpoints from the original grid; the remaining breakpoints are never repositioned. Even if all merge decisions are correct, each resulting breakpoint is therefore an element of the grid and can be off by up to $d-1$ indices from its optimal position, which is why the breakpoints are subsequently refined. Each breakpoint is moved by at most a quarter of the length of its adjacent segments, which keeps the search local and thus cheap. Formally, for consecutive breakpoints $b_{i-1} < b_i < b_{i+1}$ with adjacent segment widths $\delta_l \coloneqq b_i - b_{i-1}$ on the left and $\delta_r \coloneqq b_{i+1} - b_i$ on the right, the fine-tuning set for b_i is given by

$$\mathbb{F}(b_i;\, b_{i-1}, b_{i+1}) \coloneqq \left\{ b \in \mathbb{N} \;\middle|\; \max\left(b_i - \tfrac{\delta_l}{4},\, b_{i-1} + d\right) \leq b \leq \min\left(b_i + \tfrac{\delta_r}{4},\, b_{i+1} - d\right) \right\}, \tag{6}$$

i.e., it contains all candidates within a quarter of the adjacent segments of b_i that leave at least d points on either side, as required for a well-posed linear fit. A fine-tuning step for b_i replaces the breakpoint b_i by

$$\underset{b \,\in\, \mathbb{F}(b_i;\, b_{i-1}, b_{i+1})}{\arg\min} \; \text{RSS}(b_{i-1}, b) + \text{RSS}(b, b_{i+1}). \tag{7}$$

Algorithm 1 Fine-Tuning of Breakpoints (Lößer et al., 2025)

function FINETUNE($\mathbb{B}$, $\boldsymbol{X}$, $\boldsymbol{y}$)
  Denote $\mathbb{B} = (b_1 = 1 < b_2 < \cdots < b_{|\mathbb{B}|} = N + 1)$
  For each interior index $i \in \{2, \ldots, |\mathbb{B}| - 1\}$, let $\delta_i \leftarrow \max(b_i - b_{i-1},\ b_{i+1} - b_i)$
  Let $(i_1, \ldots, i_{|\mathbb{B}|-2})$ be the interior indices sorted by δ_i in descending order
  for $i \in (i_1, \ldots, i_{|\mathbb{B}|-2})$ **do**
    $b_i \leftarrow \underset{b \in \mathbb{F}(b_i;\, b_{i-1}, b_{i+1})}{\arg\min} \Big[\mathrm{RSS}(b_{i-1}, b) + \mathrm{RSS}(b, b_{i+1}) \Big]$ ▷ see Equations (3) and (6)
  return $\mathbb{B} = (b_1 < b_2 < \cdots < b_{|\mathbb{B}|})$

In a single sweep, breakpoints are updated sequentially rather than simultaneously, in decreasing order of their maximal adjacent segment sizes $\delta = max(\delta_l, \delta_r)$, so that each update already accounts for the previously moved neighbours. Algorithm 1 gives the corresponding pseudocode.

Complexity. In total, MERGE+FT runs in $\mathcal{O}(N \log N + N \cdot d^2)$ time and requires $\mathcal{O}(N \cdot d)$ memory (Lößer et al., 2025). The key ingredient is that, using precomputed prefix sums, $\mathrm{RSS}(b_i, b_j)$ can be evaluated in time independent of the segment length. Appendix C provides further details and the pseudocode of MERGE+FT.

3.2 GREEDY SPLITTING AND FINE-TUNE (SPLIT+FT)

We propose SPLIT+FT, short for Greedy Splitting and Fine-Tune, which combines classical greedy splitting (Bai, 1997) with the fine-tuning step of Lößer et al. (2025), see Figure 1 again for an illustration. In contrast to greedy merging, which reaches a segmentation into k intervals after $\mathcal{O}(N)$ merges, greedy splitting needs merely $\mathcal{O}(k)$ splits. Interleaving these splits with fine-tuning compensates for the early split decisions, which are made with little information about the final segmentation.

3.2.1 PROPOSED ALGORITHM

Greedy splitting. The idea of greedy splitting is to iteratively add the breakpoint that reduces the total RSS the most. Let $b_1 = 1 < b_2 < \cdots < b_{k'} < b_{k'+1} = N + 1$ be the current split of the data into k' segments. The total RSS of the best piecewise-linear function on this segmentation is

$$\mathrm{RSS}(b_1, \ldots, b_{k'+1}) \coloneqq \mathrm{RSS}(b_1, b_2) + \cdots + \mathrm{RSS}(b_{k'}, b_{k'+1}). \tag{8}$$

For an additional breakpoint $b_i < b' < b_{i+1}$ we define the *split gain*

$$\mathrm{SplitGain}(b'; b_i, b_{i+1}) \coloneqq \mathrm{RSS}(b_i, b_{i+1}) - \mathrm{RSS}(b_i, b') - \mathrm{RSS}(b', b_{i+1}) \ \geq\ 0. \tag{9}$$

Since all segments other than the one from b_i to b_{i+1} remain unchanged, the split gain is precisely the reduction of the total RSS obtained by inserting b', i.e.,

$$\mathrm{SplitGain}(b'; b_i, b_{i+1}) = \mathrm{RSS}(b_1, \ldots, b_{k'+1}) - \mathrm{RSS}(b_1, \ldots, b_i, b', b_{i+1}, \ldots, b_{k'+1}). \tag{10}$$

It is therefore the counterpart of the merging cost used before: splitting a segment at b' decreases the total RSS by the same amount that merging the two resulting segments would add back.

In each iteration, greedy splitting refines the segmentation into one with $k' + 1$ segments by inserting the admissible breakpoint $b^\star$ into the segment from b_{i^*} to b_{i^*+1} with the largest split gain, i.e.,

$$\mathrm{SplitGain}(b^\star; b_{i^*}, b_{i^*+1}) = \max \Big\{ \mathrm{SplitGain}(b'; b_i, b_{i+1}) \ \Big|\ 1 \leq i \leq k',\ b' \in \mathbb{C}(b_i, b_{i+1}) \Big\}, \tag{11}$$

where $\mathbb{C}(b_i, b_{i+1}) \coloneqq \{b' \in \mathbb{N} \mid b_i + d \leq b' \leq b_{i+1} - d\}$ restricts the search to candidates that keep a distance of at least d to the two enclosing breakpoints. This restriction does not affect the outcome of Equation (11) in general: as multiple linear regression in d dimensions fits any d points in general position exactly and the RSS of a segment is monotone under adding data points, every excluded candidate is dominated by the admissible candidate at the corresponding segment boundary.

Interleaved Fine-tuning. Greedy splitting fixes each breakpoint when it is created and never revisits it, potentially yielding a globally suboptimal set of breakpoints, see Figure 1. We therefore fine-tune the breakpoints after every splitting step except the first, which amounts to $k - 2$ calls to Algorithm 1 in total.[3] Fine-tuning does not necessarily recover the global optimal breakpoints, but it never in-

[3] The initial breakpoint minimizes the total RSS among all admissible segmentations into two intervals and can, hence, not be improved by fine-tuning.

Algorithm 2 Greedy Splitting and Fine-Tune (SPLIT+FT)

function SPLIT+FT($\boldsymbol{X}, \boldsymbol{y}$)
  $\mathbb{B} \leftarrow (1,\ N+1)$ ▷ Ordered set of breakpoints; initially a single interval
  while $|\mathbb{B}| < k+1$ **do**
    ▷ *Phase 1: Greedy split*
    Denote $\mathbb{B} = (b_1 < b_2 < \cdots < b_{|\mathbb{B}|})$
    $(b^\star,\ g^\star) \leftarrow (_,\ 0)$
    for $l = 1, \ldots, |\mathbb{B}| - 1$ **do** ▷ Search over current segments $[b_l, b_{l+1})$
      $b \leftarrow \underset{b_l + d \le b' \le b_{l+1} - d}{\arg\max}\ \mathrm{SplitGain}(b';\ b_l, b_{l+1})$ ▷ see Equation (9)
      if $\mathrm{SplitGain}(b;\ b_l, b_{l+1}) > g^\star$ **then**
        $b^\star \leftarrow b$
        $g^\star \leftarrow \mathrm{SplitGain}(b;\ b_l, b_{l+1})$
    $\mathbb{B} \leftarrow \mathbb{B} \cup \{b^\star\}$ ▷ Insert new breakpoint (preserving order)
    ▷ Phase 2: Fine-tune all interior breakpoints
    if $|\mathbb{B}| > 3$ **then** ▷ Only when more than one breakpoint was added so far
      $\mathbb{B} \leftarrow \mathrm{FINETUNE}(\mathbb{B}, \boldsymbol{X}, \boldsymbol{y})$ ▷ see Algorithm 1
  return the least-squares fit on each interval induced by $\mathbb{B}$

creases the total RSS, as a breakpoint is only modified if this decreases the objective. Algorithm 2 shows the overall pseudocode of SPLIT+FT. A detailed description is given in Appendix D.

3.2.2 COMPLEXITY ANALYSIS

Next, we analyze the time and memory complexity of SPLIT+FT and compare it to the exact DP and to MERGE+FT (Lößer et al., 2025). For our analysis, we focus on the influence of the dataset size N given a fixed input dimension d and a fixed number k of segments. We also briefly examine the influence of the latter two parameters on the runtime.

Runtime Complexity. Following Lößer et al. (2025), we use precomputed sums to evaluate linear regressions on segments efficiently. Concretely, we store N matrices $C_1, \ldots, C_N \in \mathbb{R}^{(d+1)\times(d+1)}$, which can be computed in $\mathcal{O}(N \cdot d^2)$ time, see Appendix C for details. Given these sums, the RSS of an arbitrary segment can be obtained in $\mathcal{O}(d^3)$ time by solving the corresponding normal equations. Moving one endpoint of a segment by a single data point changes its Gram matrix by a rank-one term, so the Sherman–Morrison formula yields the updated RSS in $\mathcal{O}(d^2)$ time. Since a split gain is a sum of three RSS values (see Equation (9)), evaluating a single candidate costs $\mathcal{O}(d^3)$, whereas moving a candidate by one data point costs only $\mathcal{O}(d^2)$ (Lößer et al., 2025).

Consequently, the best split point of a segment from b_i to b_{i+1} can be found in $\mathcal{O}\big((b_{i+1}-b_i)\cdot d^2+d^3\big)$ time: the first candidate requires $\mathcal{O}(d^3)$, and each of the remaining $\mathcal{O}(b_{i+1} - b_i)$ candidates is obtained from its predecessor in $\mathcal{O}(d^2)$ time. Scanning a full segmentation into k' intervals with breakpoints $b_1 = 1 < \cdots < b_{k'+1} = N+1$ therefore takes

$$\mathcal{O}\Big(\sum_{i=1}^{k'} \big((b_{i+1}-b_i)\cdot d^2+d^3\big)\Big) = \mathcal{O}\big((b_{k'+1}-b_1)\cdot d^2+k'\cdot d^3\big) = \mathcal{O}\big(N\cdot d^2+k\cdot d^3\big) = \mathcal{O}\big(N\cdot d^2\big) \quad (12)$$

time, where we used $k' \le k$ and the fact that the candidate set $\mathbb{C}$ forces every segment to contain at least d data points, so that $k \cdot d \le N$. With $k-1$ splitting iterations, greedy splitting thus runs in $\mathcal{O}(N \cdot d^2 \cdot k)$ time. One fine-tuning step has a time complexity of $\mathcal{O}(N \cdot d^2)$ (Lößer et al., 2025); we restate the argument in Appendix E. As SPLIT+FT performs $k-2$ such steps, fine-tuning also contributes only $\mathcal{O}(N \cdot d^2 \cdot k)$ time to the overall runtime. The overall runtime complexity is therefore $\mathcal{O}(N \cdot d^2 \cdot k)$. In particular, it scales linearly in the dataset size N for fixed d and k and is independent of the breakpoint locations.

Memory Complexity. The memory complexity is dominated by the space needed to store the sums $C_1, \ldots, C_N \in \mathbb{R}^{(d+1)\times(d+1)}$, leading to $\mathcal{O}(N \cdot (d+1)^2) = \mathcal{O}(N \cdot d^2)$ memory overall.

Table 1 summarises the complexities of all three algorithms. In terms of runtime, both heuristics improve substantially on the quadratic dependence of DP on N. Compared to MERGE+FT, SPLIT+FT removes the $\log N$ factor and is thus strictly linear in N, but incurs an additional factor k due to the

Table 1: Memory and time complexity of the compared algorithms for min-ϵ segmented regression, where N denotes the number of data points, d the input dimension and k the number of segments.

Algorithm	Name	Memory	Time	Solution
Dynamic Programming	DP	$\mathcal{O}(Nk)$	$\mathcal{O}(N^2(d^2+k))$	Exact
Greedy Merging and Fine-Tune	MERGE+FT	$\mathcal{O}(Nd)$	$\mathcal{O}(N(\log N + d^2))$	Heuristic
Greedy Splitting and Fine-Tune	SPLIT+FT	$\mathcal{O}(Nd^2)$	$\mathcal{O}(Nd^2k)$	Heuristic

$k-1$ iterations. The two bounds are comparable when $d^2 \cdot (k-1) \approx \log N$, so the bound of SPLIT+FT is favourable for large datasets with a moderate number of segments and a small input dimension. All methods have memory requirements linear in N. To support an unrestricted search over breakpoint positions, SPLIT+FT stores precomputed sums at all N indices, requiring a factor of d more memory than MERGE+FT. In contrast, MERGE+FT initially uses segments of d points and departs from this segmentation only during the final fine-tuning step, so it stores these sums only at every d-th index. This additional memory enables greater flexibility in breakpoint selection, with the trade-off becoming more relevant for large d.

4 EXPERIMENTS

We compare the performance and runtime of SPLIT+FT against the exact baseline DP and the heuristic MERGE+FT (Lößer et al., 2025) on synthetic and real-world datasets.

4.1 EXPERIMENTAL SETUP

For MERGE+FT, we use the publicly available implementation of Lößer et al. (2025), which is written in C++, along with Python wrappers. SPLIT+FT and DP are both implemented in Python, but compiled to machine code using the Numba just-in-time compiler (Lam et al., 2015). All experiments were performed on an *AMD EPYC 7402P 24-Core* system running Ubuntu 24.04 LTS.

4.1.1 DATASETS

We conducted experiments on four datasets. Synthetic datasets were generated using random breakpoints and segment functions, yielding a true underlying function and noisy observations for model fitting. Here, the number N of data points, the number k of segments, the standard deviation σ of observation noise and the random seed were varied depending on the context. For a k and random seed, the underlying true function remains unchanged as N and σ vary. We considered three types:

- CONT-LIN: the true function is continuous piecewise-linear as in Lößer et al. (2025)
- DISC-LIN: the true function is discontinuous piecewise-linear
- CONT-POLY: the true function is continuous piecewise-polynomial

For these types, we created datasets with $k = 6$ segments and an observation noise standard deviation of $\sigma = 0.1$. We iteratively doubled the number of data points N from 100 to $2^{19} \cdot 100 \approx 5 \cdot 10^7$. Since DP scales quadratically in N, it was evaluated only up to $N = 25\,600$. For CONT-POLY, we considered polynomials of degree 3. Each configuration was repeated for $L = 100$ random seeds.

Following Lößer et al. (2025), we used a dataset of CPU usage recorded during the execution of the *bwa* program (Li, 2013) within the *sarek* workflow (Garcia et al., 2020). It contains 70 607 CPU activity measurements sampled every 2 s and exhibits approximately four constant stages; accordingly, we fitted all methods with $k = 4$ segments. In contrast to the synthetic data, the observation noise is non-Gaussian and its variance differs between stages.

4.1.2 EVALUATION METRICS

On the synthetic datasets, we reported the mean squared error of the fitted function $\hat{f}$ with respect to the true function f and with respect to the noisy observations $y_1, \ldots, y_N$, i.e.,

$$\mathrm{MSE}_{true} \coloneqq \frac{1}{N}\sum_{n=1}^{N}\left(\hat{f}(\boldsymbol{x}_n) - f(\boldsymbol{x}_n)\right)^2 \quad \text{and} \quad \mathrm{MSE}_{obs} \coloneqq \frac{1}{N}\sum_{n=1}^{N}\left(\hat{f}(\boldsymbol{x}_n) - y_n\right)^2. \tag{13}$$

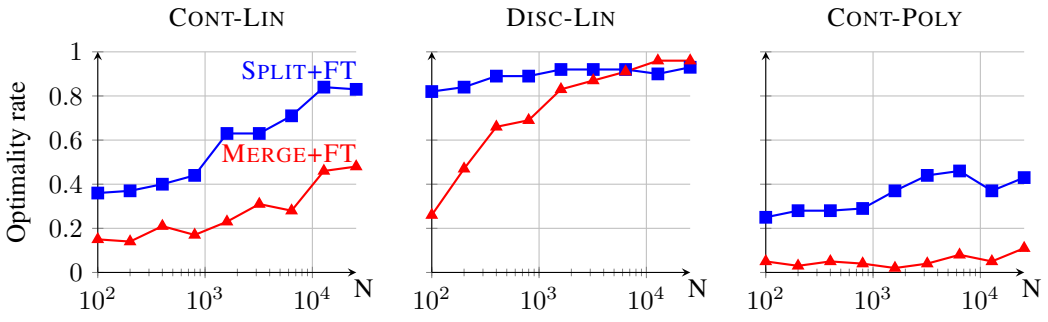

Figure 3: Optimality rate of SPLIT+FT (ours, blue squares) and MERGE+FT (red triangles) on synthetic datasets in dependence on the dataset size N.

All figures show the median over the $L = 100$ seeds together with the interquartile range, i.e. the interval between the 25% and the 75% quantile. We also report the *optimality rate* of the heuristics, i.e., the fraction of runs in which a heuristic recovers the optimal segmentation computed by DP:

$$\text{Optimality rate} := \frac{1}{L} \#\{l \in \{1, \dots, L\} \mid \mathbb{B}_l = \mathbb{B}_l^*\}. \tag{14}$$

Here, $\mathbb{B}_l$ denotes the breakpoints returned by a heuristic in the l-th run and $\mathbb{B}_l^*$ those returned by DP.

4.2 RESULTS

We first compare the predictive performance of our approach (SPLIT+FT) with the heuristic Greedy Merging and Fine-Tune approach (MERGE+FT) from Lößer et al. (2025) and the optimal Dynamic Programming (DP) approach, and then compare their runtimes. An ablation study, deferred to Appendix J, confirms that both components of our algorithm are necessary: neither greedy splitting nor fine-tuning alone matches the performance of SPLIT+FT.

4.2.1 PREDICTIVE PERFORMANCE ON SYNTHETIC DATASETS

Figure 3 visualizes the optimality rates. On the linear datasets CONT-LIN and DISC-LIN, the optimality rates of both approaches increase with increasing sample size N. On CONT-LIN, SPLIT+FT agrees with DP more than 20 percentage points more often than MERGE+FT across all dataset sizes, reaching an optimality rate of over 80% at $N = 25\,600$. On DISC-LIN, the gap is largest for small datasets: at $N = 100$, SPLIT+FT attains around 80% while MERGE+FT reaches only about 25%. For $N > 2000$, however, both methods exceed 90% and their optimality rates converge. On CONT-POLY, the optimality rate of SPLIT+FT grows slower from just above 20% at $N = 100$ to 40% at $N = 25\,600$, whereas that of MERGE+FT remains below 10% even for large datasets.

Figure 5 in Appendix F reports the corresponding MSE_{obs} and MSE_{true} values. On all datasets, MSE_{obs} of SPLIT+FT and MERGE+FT converges to the observation noise variance $\sigma^2 = 10^{-2}$, i.e. to the irreducible error, indicating that the fit neither underfits nor interpolates the noise. The error against the true function, MSE_{true}, decays polynomially in N for both heuristics on the linear datasets (a straight line on the log-log scale), whereas the trend is noisier on CONT-POLY.

We attribute the behaviour on DISC-LIN to the following mechanism. Discontinuities facilitate breakpoint detection for greedy splitting, since a true breakpoint at a jump yields a particularly high split gain and is therefore selected in an early iteration. Greedy merging, in contrast, appears to suffer from discontinuities: if two points on opposite sides of a large jump are placed in the same initial segment, merging this segment with either neighbour is expensive, so two larger segments separated by a smaller jump may be merged instead. Figure 6 in Appendix G illustrates such a case.

We see a possible reason for the noisy behavior of MSE_{true} on the polynomial dataset CONT-POLY in the fact that polynomial design matrices are poorly conditioned and both heuristics evaluates the RSS also on short segments, which may lead to numerical instabilities. Nevertheless, both heuristics are able to fit functions that fit the observed data well. A detailed investigation is left to future work.

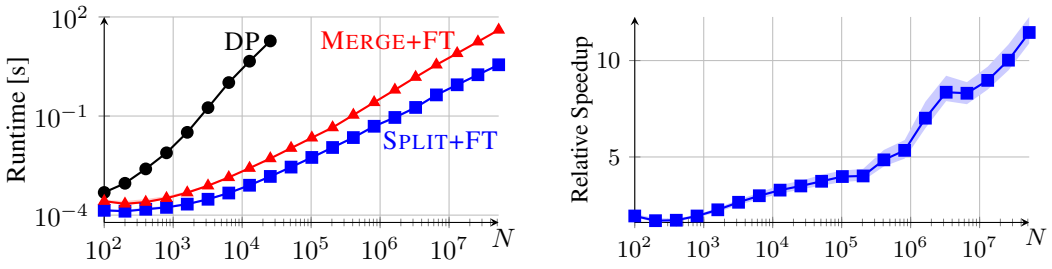

Figure 4: Left: Runtime of SPLIT+FT (ours, blue squares), MERGE+FT (red triangles) and DP (black dots) on the synthetic linear datasets (CONT-LIN and DISC-LIN) in dependence on the dataset size N. Right: Relative speedup of SPLIT+FT to MERGE+FT.

4.2.2 EXPERIMENTAL TIME COMPLEXITY ON SYNTHETIC DATASETS

We measure the runtime of all methods on the linear datasets CONT-LIN and DISC-LIN, which share the same input dimension and therefore the same runtime profile and show the results in Figure 4.

Both heuristics scale considerably better than DP: for $N \geq 10^4$, the measured runtime of SPLIT+FT grows linearly in N, in line with the $\mathcal{O}(Nd^2k)$ bound of Section 3.2.2. For smaller datasets, fixed costs such as memory allocation and terms that are asymptotically dominated still contribute noticeably, which explains the flatter slope in this regime; the same pattern is visible for MERGE+FT.

The right panel of Figure 4 reports the resulting speedup of SPLIT+FT over MERGE+FT. It grows from a factor of 2–3 at $N \approx 10^4$ to 5 at $N \approx 10^6$ and 10 at $N \approx 3 \times 10^7$.

Finally, Figures 8 and 9 in Appendix I confirm the predicted linear dependence on the number of segments k and a cubic dependence on the input dimension d (as we do not use Sherman–Morrison updates in our implementation, see Appendix D).

4.2.3 REAL-WORLD DATASET

Table 2: CPU usage results.

	DP	MERGE+FT	SPLIT+FT
Runtime (ms)	57249.4	14.9	**5.2**
MSE ($\times 10^4$)	**1.827**	**1.827**	**1.827**
Slowdown	$\times$11009.50	$\times$2.87	$\times$**1.00**

The results on the CPU usage dataset are provided in Table 2. It can be seen that both heuristics recover globally optimal segmentations found by DP (w.r.t. the MSE). Concerning the runtimes, SPLIT+FT is about three times faster than MERGE+FT and more than 10^4 times faster than DP, which is consistent with the speedups measured on synthetic data at comparable dataset sizes (Figure 4). The fitted piecewise-linear functions are visualized in Figure 7 in Appendix H.

5 CONCLUSION

In this work, we proposed Greedy Splitting and Fine-Tune (SPLIT+FT), a heuristic algorithm for min-ϵ segmented regression that combines the top-down idea of greedily splitting segments with a local fine-tuning of the breakpoints after every split as in Lößer et al. (2025). We showed that SPLIT+FT runs in $\mathcal{O}(Nd^2k)$ time and requires $\mathcal{O}(Nd^2)$ memory for datasets of size N, inputs of dimension d and k segments. In our experiments on synthetic data, SPLIT+FT recovers the optimal segmentation reliably, especially for linear datasets, while being up to an order of magnitude faster. On a real-world dataset, it recovers the optimal solution of DP with a speedup of more than 10^4. Our method is particularly well suited to settings with small d and k. Such settings are common in practice, making our method a promising alternative to existing approaches. This work focuses on min-ϵ segmented regression. Since greedy splitting progressively improves the fit, SPLIT+FT naturally extends to the min-# formulation by splitting until the residual sum of squares falls below a prescribed tolerance. We leave a systematic evaluation of this variant and an analysis of approximation guarantees to future work.

AI USE STATEMENT

In this work, we used generative AI tools to support method implementation (Greedy Splitting and Fine-Tune and DP) and aspects of the research process. Additionally, we used generative AI tools for creating or modifying scientific figures and for identifying relevant literature. We have not used generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate mathematical claims, provide key elements of proofs or assist in the writing of proofs, propose or refine hypotheses, design research methodology or experiments, assist with translation, or support qualitative and thematic data analysis. Dataset cleaning and reformatting datasets is not applicable to this work. We have reviewed all AI-assisted work. In particular, the first author checked the LLM-generated code behaves as described in the manuscript. For DP, results on linear datasets were tested against existing implementations of Dynamic Programming. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All results of our work can be reproduced using the Python code in the supplemental material. A guide to run the experiments is added in the README.md file. Upon acceptance, all code will be published in a public GitHub repository.

REFERENCES

Jayadev Acharya, Ilias Diakonikolas, Jerry Li, and Ludwig Schmidt. Fast algorithms for segmented regression. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2878–2886, New York, New York, USA, 20–22 Jun 2016a. PMLR. URL https://proceedings.mlr.press/v48/acharya16.html.

Jayadev Acharya, Ilias Diakonikolas, Jerry Li, and Ludwig Schmidt. Fast algorithms for segmented regression, 2016b. URL https://arxiv.org/abs/1607.03990.

Jushan Bai. Estimating multiple breaks one at a time. *Econometric theory*, 13(3):315–352, 1997.

Jushan Bai and Pierre Perron. Computation and analysis of multiple structural change models. *Journal of applied econometrics*, 18(1):1–22, 2003.

Rogier de Jong, Jan Verbesselt, Achim Zeileis, and Michael Schaepman. Shifts in global vegetation activity trends. *Remote Sensing*, 5(3):1117–1133, 2013.

Ben DeVries, Mathieu Decuyper, Jan Verbesselt, Achim Zeileis, Martin Herold, and Shijo Joseph. Tracking disturbance-regrowth dynamics in tropical forests using structural change detection and landsat time series. *Remote Sensing of Environment*, 169:320 – 334, 2015.

David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973.

Piotr Fryzlewicz. Wild binary segmentation for multiple change-point detection. *The Annals of Statistics*, 42(6):2243 – 2281, 2014. doi: 10.1214/14-AOS1245. URL https://doi.org/10.1214/14-AOS1245.

Maxime Garcia, Szilveszter Juhos, Malin Larsson, Pall I Olason, Marcel Martin, Jesper Eisfeldt, Sebastian DiLorenzo, Johanna Sandgren, Teresita Díaz De Ståhl, Philip Ewels, et al. Sarek: A portable workflow for whole-genome sequencing analysis of germline and somatic variants. *F1000Research*, 9:63, 2020.

Fabian Gieseke, Sabina Rosca, Troels Henriksen, Jan Verbesselt, and Cosmin Eugen Oancea. Massively-parallel change detection for satellite time series data with missing values. In *Proceedings of the 36th IEEE International Conference on Data Engineering (ICDE)*, pp. 385–396, Dallas, USA, 2020. doi: 10.1109/ICDE48307.2020.00040.

James W. Hawkins, Barend M. de C. Bronsvoort, Wilfred O. Odadi, Endale B. Gurmu, Edward H. Cabezas-Garcia, Mohamed Habibou Assouma, Luc Hippolyte Dossa, Sonja M. Leitner, and Claudia Arndt. Heart girth-based bodyweight prediction for sub-saharan african ruminants: updated equations by species, breed, and system. *Veterinary and Animal Science*, 33:100724, 2026. ISSN 2451-943X. doi: 10.1016/j.vas.2026.100724. URL https://www.sciencedirect.com/science/article/pii/S2451943X26001547.

Samuel Hislop, Simon Jones, Mariela Soto-Berelov, Andrew Skidmore, Andrew Haywood, and Trung H Nguyen. A fusion approach to forest disturbance mapping using time series ensemble techniques. *Remote sensing of environment*, 221:188–197, 2019.

Eamonn Keogh. Fast similarity search in the presence of longitudinal scaling in time series databases. In *Proceedings Ninth IEEE International Conference on Tools with Artificial Intelligence*, pp. 578–584. IEEE, 1997.

Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. An online algorithm for segmenting time series. In *Proceedings 2001 IEEE international conference on data mining*, pp. 289–296. IEEE, 2001.

Rebecca Killick, Paul Fearnhead, and Idris A Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, 2012.

Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6, 2015.

Heng Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem, 2013. URL https://arxiv.org/abs/1303.3997.

Ansgar Lößer, Max Schlecht, Florian Schintke, Joel Witzke, Matthias Weidlich, and Björn Scheuermann. Fast min-ϵ segmented regression using constant-time segment merging. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 40312–40327. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/losser25a.html.

Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256, 1972.

Guillem Rigaill. A pruned dynamic programming algorithm to recover the best segmentations with 1 to k_{max} change-points. *Journal de la société française de statistique*, 156(4):180–205, 2015. URL http://geodesic.mathdoc.fr/item/JSFS_2015__156_4_180_0/.

Quanxi Shao and N. A. Campbell. Applications: Modelling trends in groundwater levels by segmented regression with constraints. *Australian & New Zealand Journal of Statistics*, 44(2): 129–141, 2002. doi: 10.1111/1467-842X.00216. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-842X.00216.

A. R. Tomé and P. M. A. Miranda. Piecewise linear fitting and trend changing points of climate parameters. *Geophysical Research Letters*, 31(2), 2004. doi: 10.1029/2003GL019100. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2003GL019100.

Tobias Wies, Sami Kharma, Tobias Meuser, and Björn Scheuermann. Accurate synthetic tasks for scientific workflow benchmarks. In *2026 IEEE 26th International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW), Doctoral Symposium*, pp. 9–12, 2026.

Yohei Yamamoto and Pierre Perron. Estimating and testing multiple structural changes in linear models using band spectral regressions. *The Econometrics Journal*, 16(3):400–429, 2013.

A Piecewise-Polynomial Functions

We define piecewise-polynomial functions and show that there is a correspondence with piecewise-linear functions in higher dimensions. Note that this is the analog of the fact that polynomial functions are linear functions in higher dimensions. Recall that the polynomial encoding $\mathrm{poly}_{\text{d-1}} : \mathbb{R} \to \mathbb{R}^d$ is defined by $x \mapsto (1, x, \ldots, x^{d-1})$.

Definition 1. A function $g : \mathbb{R} \to \mathbb{R}$ is called piecewise-polynomial of degree at most $d-1$ with at most k segments if there is a partition $\mathbb{R} = \cup_{i=1}^{k} \mathbb{I}_i$ into pairwise disjoint intervals $\mathbb{I}_i \subset \mathbb{R}$, such that g is polynomial of degree as most $d-1$ on $\mathbb{I}_i$ for all $i \in \{1, \ldots, k\}$, i.e. there exist weights $w_0^{(i)}, \ldots, w_{d-1}^{(i)} \in \mathbb{R}$ such that $g(x) = \sum_{l=0}^{d-1} w_l^{(i)} \cdot x^l$ for $x \in \mathbb{I}_i$.

Lemma 1. *The map*

$$\mathbb{L}_{k,2}^{d} \to \{g : \mathbb{R} \to \mathbb{R} \mid g \textit{ is piecewise polynomial of degree at most } d-1 \textit{ with at most } k \textit{ segments}\}$$
$$f \mapsto f \circ \mathrm{poly}_{\text{d-1}}$$

is well-defined and surjective, i.e. $g : \mathbb{R} \to \mathbb{R}$ is piecewise-polynomial of degree at most $d-1$ with at most k segments if and only if there exists $f \in \mathbb{L}_{k,2}^{d}$ such that $g = f \circ \mathrm{poly}_{\text{d-1}}$. The splitting coordinate $j = 2$ comes from $pr_2 \circ \mathrm{poly}_{\text{d-1}}(x) = x$ for all $x \in \mathbb{R}$.

Proof. The general idea is that the local polynomial weights of a piecewise-polynomial function form the weight vector of a piecewise-linear function in higher dimensions via the connection $\boldsymbol{w}_i = (\boldsymbol{w}_0^{(i)}, \ldots, \boldsymbol{w}_{d-1}^{(i)}) \in \mathbb{R}^d$. Below, we write this down more explicitly.

Well-defined. Consider $f \in \mathbb{L}_{k,2}^{d}$. By definition, there is a partition of the real line into k pairwise disjoint intervals $\mathbb{I}_1, \ldots, \mathbb{I}_k \subset \mathbb{R}$ such that for every $i \in \{1, \ldots, k\}$ there exists a parameter vector $\boldsymbol{w}_i = (w_0^{(i)}, \ldots, w_{d-1}^{(i)}) \in \mathbb{R}^d$ such that $f(\boldsymbol{x}) = \boldsymbol{w}_i^{\mathrm{T}} \cdot \boldsymbol{x}$ for all inputs $\boldsymbol{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$ whose 2nd coordinate satisfies $x_2 \in \mathbb{I}_i$. Then for $x \in \mathbb{I}_i$,

$$f \circ \mathrm{poly}_{\text{d-1}}(x) = f((1, x, \ldots, x^{d-1})^{\mathrm{T}}) = \boldsymbol{w}_i^{\mathrm{T}} \cdot (1, x, \ldots, x^{d-1})^{\mathrm{T}} = \sum_{l=0}^{d-1} w_l^{(i)} \cdot x^l$$

as $pr_2 \circ \mathrm{poly}_{\text{d-1}}(x) = x$ and thus, $f \circ \mathrm{poly}_{\text{d-1}}$ is piecewise-polynomial of degree at most $d-1$ with at most k segments by definition.

Surjectivity. Let $g : \mathbb{R} \to \mathbb{R}$ be a piecewise-polynomial function of degree at most $d-1$ with at most k segments. By definition, there is a partition $\mathbb{R} = \cup_{i=1}^{k} \mathbb{I}_i$ into pairwise disjoint intervals $\mathbb{I}_i \subset \mathbb{R}$, such that there exist weights $w_0^{(i)}, \ldots, w_{d-1}^{(i)} \in \mathbb{R}$ such that $g(x) = \sum_{l=0}^{d-1} w_l^{(i)} \cdot x^l$ for $x \in \mathbb{I}_i$. By writing $\boldsymbol{w}_i = (\boldsymbol{w}_0^{(i)}, \ldots, \boldsymbol{w}_{d-1}^{(i)}) \in \mathbb{R}^d$ we can define $f : \mathbb{R}^d \to \mathbb{R}$ by the rule $\boldsymbol{x} \mapsto \boldsymbol{w}_i^{\mathrm{T}} \cdot \boldsymbol{x}$ if $x_2 \in \mathbb{I}_i$. As the intervals $\mathbb{I}_i$ form a pairwise disjoint cover of $\mathbb{R}$, f is well-defined. Then f is a piecewise-linear function with at most k segments with respect to coordinate $j = 2$ by definition. Using a calculation as above, it can be seen that $f \circ \mathrm{poly}_{\text{d-1}} = g$. □

B Dynamic Programming

The dynamic program to solve the min-ϵ task exactly is well-known and described in detail in Acharya et al. (2016b). The key idea is to generate the matrix $\boldsymbol{A} \in \mathbb{R}^{N \times k}$, where $\boldsymbol{A}_{n,i}$ denotes the minimal error that a piecewise-linear function with $i \in \{1, \ldots, k\}$ segments can produce on the first $n \in \{1, \ldots, N\}$ data points, i.e.,

$$\boldsymbol{A}_{n,i} \coloneqq \min_{f \in \mathbb{L}_i^d} \mathrm{RSS}(f, \boldsymbol{X}_{:n+1,:}, y_{:n+1}). \tag{15}$$

Note that the entry $\boldsymbol{A}_{n,i}$ is zero for $n \leq i \cdot d$ in general, since multiple linear regression in d dimensions is able to exactly fit d points in general position. Furthermore, there is a relationship between the entries of $\boldsymbol{A}$: the best split using i segments is determined by the best combination of a piecewise-linear function using $i-1$ segments on the first part of the dataset and the residual sum of

squared errors of a linear regression on the last part of the dataset. In other words, the relationship is giving by

$$\boldsymbol{A}_{n,i} = \min_{n'<n} (\boldsymbol{A}_{n',i-1} + \mathrm{RSS}(n'+1, n+1)) \tag{16}$$

for $i > 1$ and $n > 1$. This can be used to iteratively fill out the matrix $\boldsymbol{A}$ and deduce the best piecewise-linear function from it. It is well known that the exact dynamic program runs in time $\mathcal{O}(N^2 \cdot (d^2 + k))$ (Acharya et al., 2016b, Theorem 15.) and has a memory complexity in $\mathcal{O}(N \cdot k)$ due to storing $\boldsymbol{A}$.

C Efficient Computation of the RSS via Prefix Sums

Let $1 \leq b < b' \leq N + 1$ be two breakpoints.

Recall that computing $\mathrm{RSS}(b, b')$ requires the least-squares solution

$$\beta \coloneqq (\boldsymbol{X}_{b:b'}^{\mathrm{T}} \cdot \boldsymbol{X}_{b:b'})^{-1} \cdot \boldsymbol{X}_{b:b'}^{\mathrm{T}} \cdot \boldsymbol{y}_{b:b'} \in \mathbb{R}^d \tag{17}$$

on the segment ranging from index b (included) to index b' (excluded). A key observation in Acharya et al. (2016a) and Lößer et al. (2025) is that the formation of the appearing matrices

$$A_{b:b'} \coloneqq \boldsymbol{X}_{b:b'}^{\mathrm{T}} \cdot \boldsymbol{X}_{b:b'} \in \mathbb{R}^{d \times d}$$
$$B_{b:b'} \coloneqq \boldsymbol{X}_{b:b'}^{\mathrm{T}} \cdot \ \boldsymbol{y}_{b:b'} \in \mathbb{R}^d$$

is additive with respect to interval extension: for all $b < b' < b''$,

$$A_{b:b''} = A_{b:b'} + A_{b':b''}$$
$$B_{b:b''} = B_{b:b'} + B_{b':b''}.$$

Exploiting this additivity, $A_b \coloneqq A_{1:b} \in \mathbb{R}^{d \times d}$, $B_b \coloneqq B_{1:b} \in \mathbb{R}^d$ and $s_b \coloneqq \sum_{i=1}^{b-1} y_i^2 \in \mathbb{R}$ are stored as prefix sums. The statistics of an arbitrary segment are then obtained as differences

$$A_{b:b'} = A_{b'} - A_b \in \mathbb{R}^{d \times d}$$
$$B_{b:b'} = B_{b'} - B_b \in \mathbb{R}^d$$
$$s_{b:b'} = s_{b'} - s_b \in \mathbb{R},$$

which costs $\mathcal{O}(d^2)$ operations, independently of the segment length $b' - b$.

Since the prefix sums satisfy the recurrences

$$A_{b+1} = A_b + \boldsymbol{x}_b \cdot \boldsymbol{x}_b^{\mathrm{T}} \in \mathbb{R}^{d \times d}$$
$$B_{b+1} = B_b + \boldsymbol{x}_b \cdot y_b \in \mathbb{R}^d$$
$$s_{b+1} = s_b + y_b^2 \in \mathbb{R},$$

they can all be precomputed in a single pass over the dataset in $\mathcal{O}(Nd^2)$ time, each step being dominated by the rank-one update of A. Storing them requires $\mathcal{O}(Nd^2)$ memory, which is the dominant memory cost of SPLIT+FT.[4]

The following steps yield an efficient way of calculating $\mathrm{RSS}(b, b')$: To keep the notation light, we drop the segment indices in this derivation and write $\boldsymbol{X} \coloneqq \boldsymbol{X}_{b:b'}$, $\boldsymbol{y} \coloneqq \boldsymbol{y}_{b:b'}$, $A \coloneqq A_{b:b'}$, $B \coloneqq B_{b:b'}$, $s \coloneqq s_{b:b'}$ and $\beta \coloneqq \beta_{b:b'}$. Expanding the squared norm and inserting $\beta = A^{-1}B$ gives

$$\begin{aligned}
\mathrm{RSS}(b, b') &= \left\|\boldsymbol{y} - \boldsymbol{X}\beta\right\|_2^2 = (\boldsymbol{y} - \boldsymbol{X}\beta)^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\beta) \\
&= \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\beta - \beta^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} + \beta^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\beta \\
&= \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - 2\,\beta^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} + \beta^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\beta && (\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\beta \in \mathbb{R} \text{ is a scalar}) \\
&= s - 2\,\beta^{\mathrm{T}}B + \beta^{\mathrm{T}}A\beta && (\text{definition of } A, B, s) \\
&= s - 2\,B^{\mathrm{T}}A^{-1}B + B^{\mathrm{T}}A^{-1}AA^{-1}B && (\text{inserting } \beta = A^{-1}B,\ A^{-1} \text{ symmetric}) \\
&= s - B^{\mathrm{T}}A^{-1}B. && (18)
\end{aligned}$$

[4] As A_b is symmetric, it suffices to store its $d(d+1)/2$ upper-triangular entries, which halves the constant but not the asymptotic memory requirement.

Algorithm 3 Greedy Merging and Fine-Tune (Lößer et al., 2025).

function MERGE+FT($\boldsymbol{X}, \boldsymbol{y}$)
  ▷ *Phase 1: Initial Placement*
  $\mathbb{B} \leftarrow \big((m-1)\,d + 1 : m = 1, \dots, \lfloor N/d \rfloor\big) \cup (N+1)$ ▷ Uniform initial grid
  ▷ *Phase 2: Greedy Merging*
  while $|\mathbb{B}| > k + 1$ **do**
    $l^\star \leftarrow \underset{2 \le l \le |\mathbb{B}|-1}{\arg\min} \big[\mathrm{RSS}(b_{l-1}, b_{l+1}) - \mathrm{RSS}(b_{l-1}, b_l) - \mathrm{RSS}(b_l, b_{l+1})\big]$
    $\mathbb{B} \leftarrow \mathbb{B} \setminus \{b_{l^\star}\}$ ▷ Merge the two adjacent segments with smallest cost
  ▷ *Phase 3: Fine-Tuning*
  $\mathbb{B} \leftarrow$ FINETUNE($\mathbb{B}, \boldsymbol{X}, \boldsymbol{y}$)
  return the least-squares fit on each interval induced by $\mathbb{B}$

i.e. reinstating the segment indices, the RSS of a segment is fully determined by its three sufficient statistics $A_{b:b'}$, $B_{b:b'}$ and $s_{b:b'}$, and hence by the prefix sums at the two endpoints. As this involves inverting the matrix $A_{b:b'} \in \mathbb{R}^{d \times d}$, the runtime is in $\mathcal{O}(d^3)$.

Extending a segment by a single data point is considerably cheaper. Recall that

$$A_{b:b'+1} = A_{b:b'} + \boldsymbol{x}_{b'} \cdot \boldsymbol{x}_{b'}^{\mathrm{T}}$$

is a rank-1-update of $A_{b:b'}$. Hence, if $A_{b:b'}^{-1}$ is already known, the Sherman–Morrison formula

$$\big(A + uu^{\mathrm{T}}\big)^{-1} = A^{-1} - \frac{A^{-1}uu^{\mathrm{T}}A^{-1}}{1 + u^{\mathrm{T}}A^{-1}u} \tag{19}$$

yields $A_{b:b'+1}^{-1}$ in $\mathcal{O}(d^2)$ time. The same holds for shrinking a segment, which corresponds to a rank-one downdate. Therefore, the RSS of a segment whose endpoint has moved by one data point is obtained in $\mathcal{O}(d^2)$ overall.

The algorithm for Greedy Merging and Fine-Tune is given in Algorithm 3.

D Implementation Details

In our implementation of SPLIT+FT, we adjusted a few things compared to the algorithm as analyzed in Section 3.2.2.

1-d Kernel. For scalar input, after adding an intercept, linear regression has $d = 2$-dimensional input. This allows to use the general formulas (as stated in Appendix C). However, there are faster ways to solve the best linear fit on a segment by using simple linear regression. That is, for a segment from b (included) to b' (excluded) of length $m \coloneqq b' - b$, with segment means

$$\bar{x} \coloneqq \frac{1}{m}\sum_{i=b}^{b'-1} x_i \in \mathbb{R}, \qquad \bar{y} \coloneqq \frac{1}{m}\sum_{i=b}^{b'-1} y_i \in \mathbb{R},$$

and centred (co-)variances

$$\begin{aligned}
S_{xx} &\coloneqq \sum_{i=b}^{b'-1}(x_i - \bar{x})^2 = \sum_{i=b}^{b'-1} x_i^2 - m\,\bar{x}^2 \in \mathbb{R},\\
S_{yy} &\coloneqq \sum_{i=b}^{b'-1}(y_i - \bar{y})^2 = \sum_{i=b}^{b'-1} y_i^2 - m\,\bar{y}^2 \in \mathbb{R},\\
S_{xy} &\coloneqq \sum_{i=b}^{b'-1}(x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=b}^{b'-1} x_i y_i - m\,\bar{x}\,\bar{y} \in \mathbb{R},
\end{aligned}$$

the best fitting slope $s \in \mathbb{R}$ and intercept $a \in \mathbb{R}$ are

$$s = \frac{S_{xy}}{S_{xx}} \in \mathbb{R}, \qquad a = \bar{y} - s\,\bar{x} \in \mathbb{R}, \tag{20}$$

and the corresponding residual sum of squares is

$$\mathrm{RSS}(b, b') = S_{yy} - \frac{S_{xy}^2}{S_{xx}} = S_{yy} - s\, S_{xy} \in [0, \infty). \tag{21}$$

In this case, it is enough to save the prefix sums

$$P_j^x \coloneqq \sum_{i=1}^{j-1} x_i \in \mathbb{R}, \qquad P_j^y \coloneqq \sum_{i=1}^{j-1} y_i \in \mathbb{R}, \qquad P_j^{xx} \coloneqq \sum_{i=1}^{j-1} x_i^2 \in \mathbb{R},$$
$$P_j^{yy} \coloneqq \sum_{i=1}^{j-1} y_i^2 \in \mathbb{R}, \qquad P_j^{xy} \coloneqq \sum_{i=1}^{j-1} x_i y_i \in \mathbb{R}, \qquad \text{for } j = 1, \ldots, N+1,$$

Precomputing the five prefix sums takes $\mathcal{O}(N)$ time and memory. This is what is done in our implementation. Note that it does not change the asymptotic behaviour of the runtime; it only removes the constants hidden in the d^2 and d^3 terms for $d = 2$.

Sherman-Morrison Formula. The runtime analysis in Section 3.2.2 exploits that moving a candidate breakpoint by one data point changes the Gram matrix of the affected segment by a rank-one term, so that the Sherman–Morrison formula yields the updated inverse — and hence the updated RSS — in $\mathcal{O}(d^2)$ instead of $\mathcal{O}(d^3)$ time. Our implementation does not make use of this. Since we work with polynomial features, the design matrices are ill-conditioned and lead to numerical instabilities.

We therefore recompute the fit of every candidate segment from scratch: the required sums are obtained from the precomputed prefix sums, and the associated normal equations are solved via a Cholesky factorisation, without ever forming the inverse explicitly. This costs $\mathcal{O}(d^3)$ per candidate and thus raises the overall bound by one factor d; the dependence on the dataset size N and the number of segments k, which is the focus of our analysis, remains unchanged.

E Runtime Complexity of Fine-Tuning

In this section, we discuss the runtime complexity of Fine-Tuning (Algorithm 1) in $\mathcal{O}(N \cdot d^2)$ in more detail. Let k' be the current number of segments.

Lines 2-4: Please note that Lines 2 - 4 run in time $\mathcal{O}(k' \log(k'))$ using classical sort algorithms. For large k' such that $k' \log(k')) > N$, this can be reduced using counting sort:

Even if $\mathbb{B} \subset \{1, \ldots, N+1\}$ is not sorted, using counting sort, $\mathbb{B}$ can be sorted in $\mathcal{O}(N + k') = \mathcal{O}(N)$ time. Calculation of all δ_i take $\mathcal{O}(k')$ time. As all $\delta_i \in \{1, \ldots, N\}$, it can be sorted in $\mathcal{O}(N + k') = \mathcal{O}(N)$ time, again using counting sort.

Lines 5 - 6: For one i, the calculation of the first $\mathrm{RSS}(b_{i-1}, b) + \mathrm{RSS}(b, b_{i+1})$ costs $\mathcal{O}(d^3)$ time and the following evaluations only $\mathcal{O}(d^2)$ using the Sherman-Morrison formula, see Appendix C. This results in a runtime in $\mathcal{O}(d^3 + \frac{\delta_i}{2} d^2)$. Therefore, both lines together need

$$\mathcal{O}(\sum_{i=2}^{k'} (d^3 + \tfrac{\delta_i}{2} d^2)) = \mathcal{O}(k' d^3 + \sum_{i=2}^{k'} \tfrac{\delta_i}{2} d^2) = \mathcal{O}(k' d^3 + N d^2) = \mathcal{O}(N d^2),$$

where we used $\sum_{i=2}^{k'} \frac{\delta_i}{2} \leq N$ in the second equation and that all segments have at least d elements, i.e. $k' \cdot d \leq N$, in the third equation.

Altogether, the Fine-Tuning algorithm runs in $\mathcal{O}(N d^2)$ time.

F Additional Performance Results

In Figure 5, we present the performance of the compared methods on the synthetic datasets.

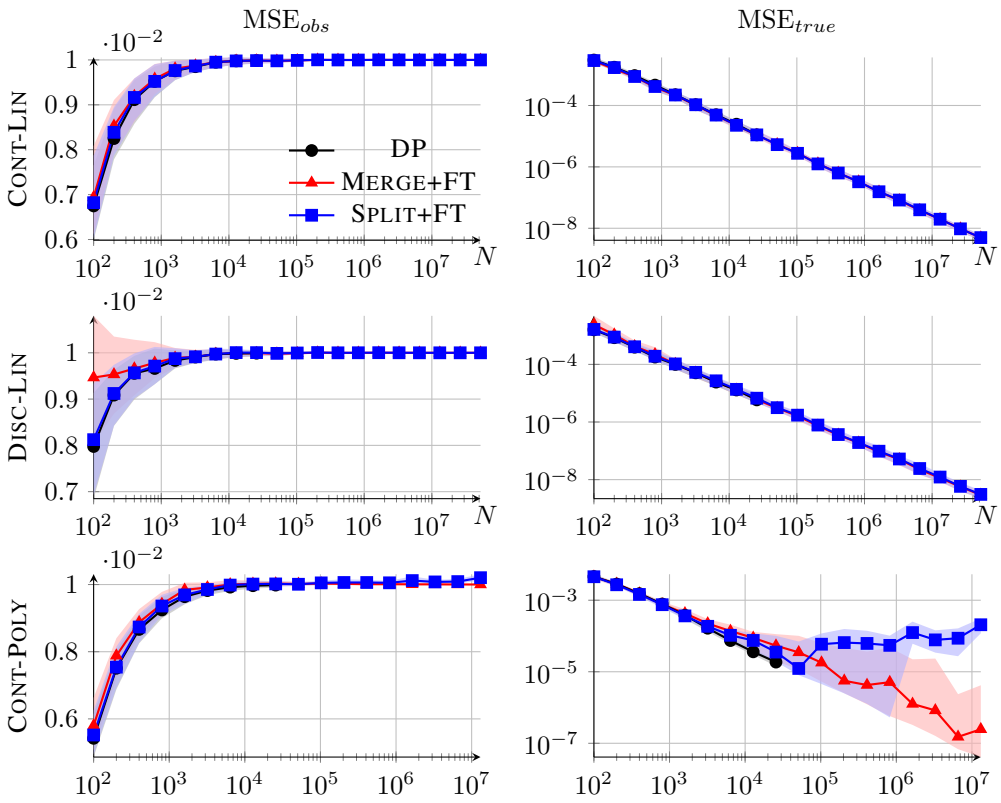

Figure 5: Metrics (Left: MSE$_{obs}$; Right: MSE$_{true}$) on the synthetic datasets Cont-Lin (first row), Disc-Lin (second row) and Cont-Poly (third row) in dependence of the sample size N.

G Exemplary Predictions on Disc-Lin

In Figure 6, predictions for an exemplary Disc-Lin dataset of length $N = 100$ are visualized.

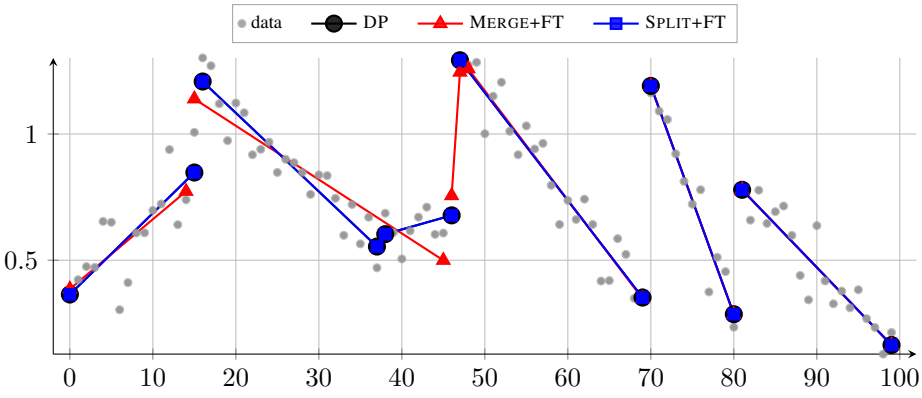

Figure 6: The Disc-Lin dataset of length $N = 100$ (seed $= 1$) together with the predictions of Split+FT (ours, blue squares), Merge+FT (red) and DP (black dots). Split+FT's predictions coincide with DP, while Merge+FT suffers from the high jump between index 46 and 47, which form its third segment. The merging cost for adjacent segments was so high that it was never merged.

H CPU USAGE RESULTS

The CPU usage dataset and predictions are presented in Figure 7.

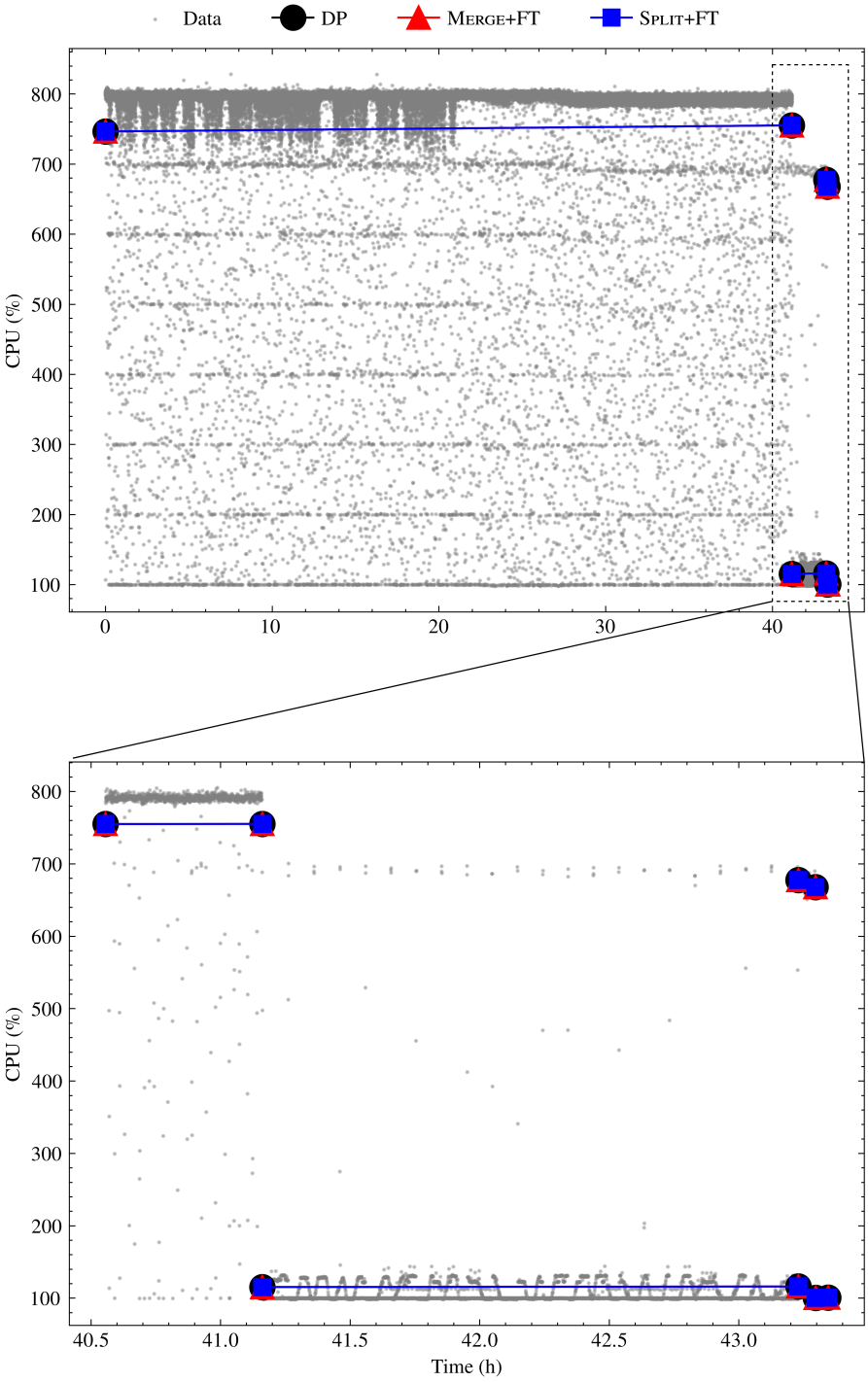

Figure 7: CPU usage dataset together with the predictions of SPLIT+FT (ours, blue squares), MERGE+FT (red triangles) and DP (black dots). All method return the same predictions. Top: Full dataset, Bottom: Zoomed in on the last three hours.

I RUNTIME BEHAVIOR WITH RESPECT TO NUMBER OF SEGMENTS AND INPUT DIMENSION

In this section, we present runtime results for varying number of segments k (Figure 8) and for varying input dimension d (Figure 9).

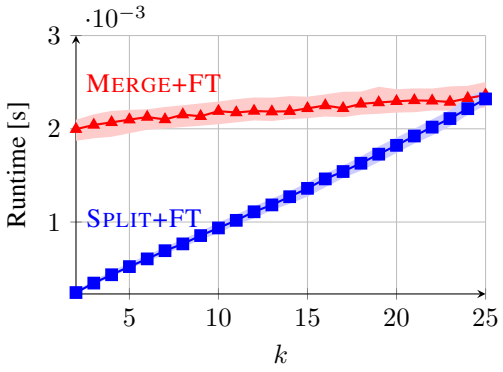

Figure 8: Runtime of SPLIT+FT (ours, blue squares) and MERGE+FT (red triangles) on CONT-LIN of size $N = 10^5$ in dependence on the number of segment k.

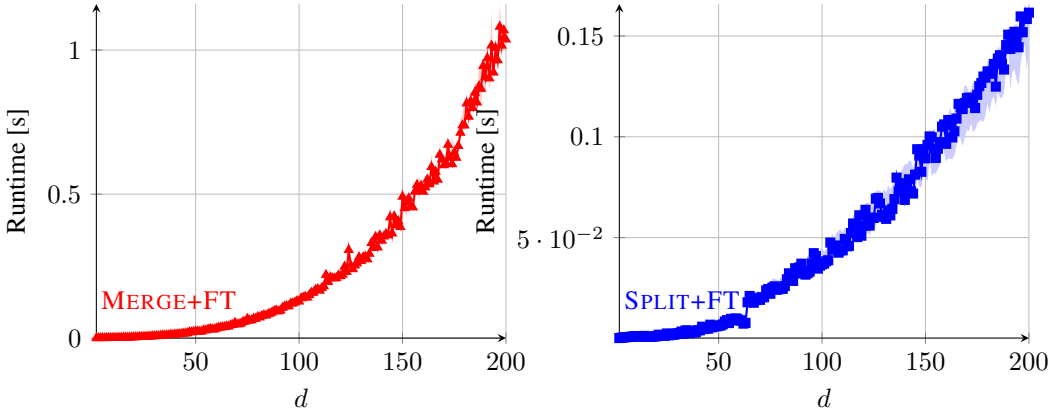

Figure 9: Runtime of SPLIT+FT (ours, blue squares) and MERGE+FT (red triangles) on CONT-LIN of size $N = 10^4$ in dependence of the input dimension d, i.e. the polynomial degree $d - 1$.

J ABLATION STUDY RESULTS

To separate the contributions of the two components of Greedy Splitting and Fine-Tune, we perform an ablation over a 2×2 design: the splitting rule (random or greedy) and the fine-tuning step (absent or present). This yields the following four variants:

- RS (Random Splitting): in each iteration, a breakpoint is drawn uniformly at random from the admissible candidates.
- RS+FT (Random Splitting and Fine-Tune): as RS, but all breakpoints are fine-tuned after every splitting step.
- GS (Greedy Splitting): in each iteration, the admissible breakpoint with the highest split gain is inserted, i.e. Greedy Splitting and Fine-Tune without fine-tuning.
- SPLIT+FT (Greedy Splitting and Fine-Tune): our proposed algorithm, combining greedy splitting with fine-tuning after every splitting step.

Comparing RS to GS (and RS+FT to SPLIT+FT) isolates the effect of the greedy split criterion, whereas comparing GS to SPLIT+FT (and RS to RS+FT) isolates the effect of fine-tuning. Using

the same experimental setup as before in Section 4.1, we compare the performance and runtime of the resulting four model variants.

Approximation Quality. Figure 10 reports MSE_{obs} and MSE_{true} for all four variants. Across all three datasets, Greedy Splitting and Fine-Tune achieves the lowest MSE_{true}, and the gap widens with the dataset size.

On CONT-LIN, RS+FT recovers the true function slightly better than GS, but its performance plateaus for large datasets, whereas SPLIT+FT keeps improving as more data become available. On DISC-LIN, GS outperforms RS+FT and nearly matches SPLIT+FT in terms of MSE_{obs}, yet remains clearly worse in terms of MSE_{true}. On CONT-POLY, GS and RS+FT perform comparably, and both are outperformed by SPLIT+FT.

The discrepancy between the two metrics on DISC-LIN is instructive: a segmentation can fit the observations about as well as SPLIT+FT while placing its breakpoints noticeably worse. Fine-tuning therefore does not merely reduce the training objective, it improves the localisation of the breakpoints, which is what MSE_{true} measures. This also shows that the two components are complementary rather than interchangeable: neither greedy splitting nor fine-tuning alone matches the combination.

Runtime. Figure 11 shows the corresponding runtimes. On the linear datasets, adding fine-tuning and switching from random to greedy splitting incur overheads of comparable magnitude. In contrast, on CONT-POLY, the overhead of greedy splitting is much smaller than the overhead of fine-tuning.

Overall, the accuracy gains of SPLIT+FT come at a moderate cost: both components add a constant factor to the runtime and leave the linear scaling in N intact, as predicted by the complexity analysis in Section 3.2.2. Their relative cost, however, depends on the input dimension: for $d = 2$ both contribute comparably, whereas for $d = 4$ fine-tuning clearly dominates.

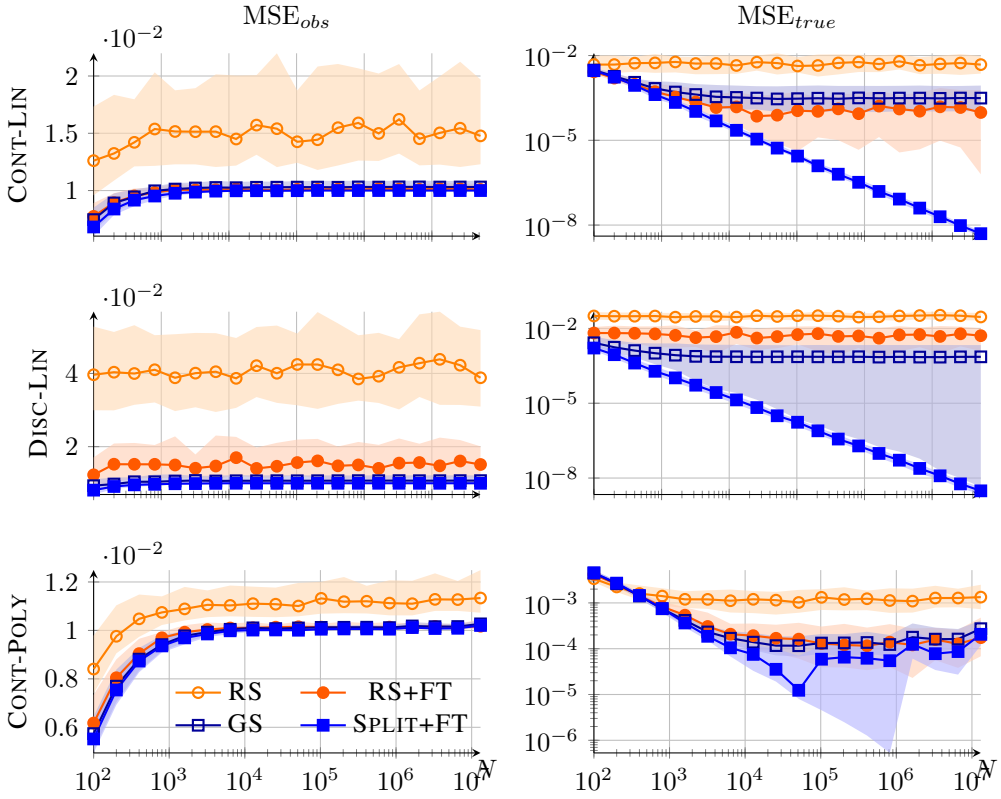

Figure 10: Metrics (Left: MSE_{obs}; Right: MSE_{true}) on the synthetic datasets CONT-LIN (first row), DISC-LIN (second row) and CONT-POLY (third row) in dependence of the sample size N for the ablation methods.

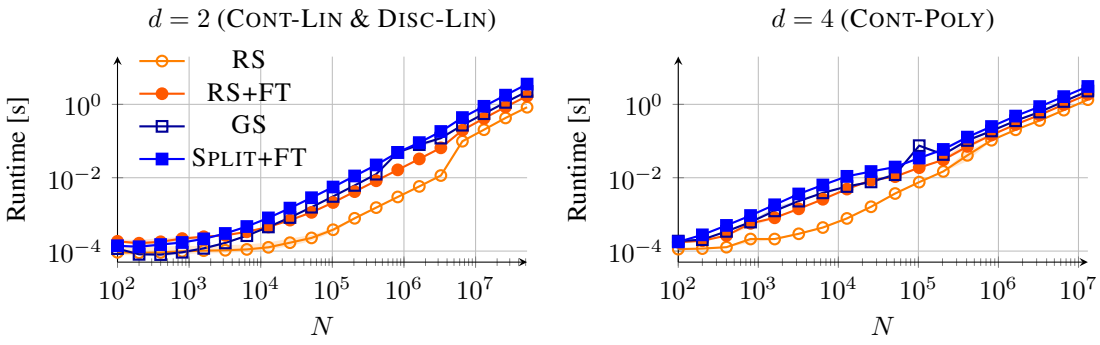

Figure 11: Runtime of ablation methods in dependence on the dataset size N. Left: synthetic linear datasets (CONT-LIN and DISC-LIN) Right: synthetic polynomial dataset CONT-POLY.