Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Shift-Safe EStream-OPF: Reusable Top-K Forgetting Queries over Order-Preserving Index Streams

Khac-Chien Nguyen
nkchienster@gmail.com
People's Police University
Nhut-Minh Bui
People's Police University
Cong-Phat Tran
People's Police University

Research Article

Keywords:

Posted Date: November 10th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11029579/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Shift-Safe EStream-OPF: Reusable Top-K Forgetting Queries over Order-Preserving Index Streams

Khac-Chien Nguyen[1*], Nhut-Minh Bui[1] and Cong-Phat Tran[1]

[1*]Faculty of Information Technology, People's Police University, Ho Chi Minh City, Vietnam.

*Corresponding author(s). E-mail(s): nkchienster@gmail.com;

Abstract

Order-preserving (OP) indexes compact time-series windows that share relative-order structure, whereas forgetting-aware OP mining has mainly been studied as threshold-driven single-shot mining. We study a build-once/query-many problem: repeated global top-$\boldsymbol{K}$ ranking over a fixed weak-order index under endpoint exponential forgetting. A direct product-form factorization is algebraically correct but can create a binary64 intermediate in which an old-occurrence mass underflows while a positive depth exponential overflows. We derive a shift-safe factorization anchored at the rightmost occurrence start, so all evaluated exponential arguments are nonpositive and the rightmost-start term is exactly one. The resulting equivalence-stream query layer, EStream-OPF, formalizes canonical weak-order identity, maximal occurrence-equivalent streams, a deterministic machine comparator, rounded-score plateaus, $\boldsymbol{K}$-head filtering, and comparator-consistent multiway merge. A Java 21 artifact implements the proposed method (B6) and five fair baselines/ablations. Across ten real benchmark instances from nine public data products, with series length $\boldsymbol{n}$ up to 500,000, B6 has lower median latency than the per-locus capacity-$\boldsymbol{K}$ heap baseline (B2) on 8/10 instances at maximum depth $\boldsymbol{L = 12}$, $\boldsymbol{K = 50}$, and half-life $\boldsymbol{n/4}$; B2/B6 ratios include 1.72× for weekly CO_2, 1.69× for SILSO daily sunspots, 1.64× for UCI Household 100k, and 1.43× for NASDAQ. Factorization-only B4 is fastest on four datasets and stays within 6% of B6 on the four rows where B6 is fastest; Household 500k is at practical parity with B2. Reusing an index raises the build-amortized B2/B6 ratio from 1.01× for one query to 1.39× for 1,000 queries. The contribution is therefore conditional: benefits depend on occurrence-equivalent compression, the ratio $\boldsymbol{K/E}$ of requested results to eligible streams, and index reuse. The $\boldsymbol{K}$-head layer is optional once the stream mass is shared.

Keywords: order-preserving pattern mining, time-series indexing, exponential forgetting, top-K query, weak order, numerical robustness, ranked streams

1 Introduction

Order-preserving (OP) analysis represents a time-series window by relative order rather than amplitude, making trend descriptions invariant to strictly monotone changes in scale and offset. This idea underlies order-preserving matching and a growing family of OP mining methods for frequent patterns, rules, approximate patterns, contrast patterns, co-occurrence patterns, outlying occurrences, and stable patterns [1–9]. A complementary line constructs OP suffix structures and related indexes that compact shared order structure and support repeated mining or comparison tasks [10, 11].

Temporal relevance creates a different dimension. OPF-Miner attaches an exponential forgetting weight to occurrence endpoints so that recent occurrences contribute more than old ones [12]. Its objective, however, remains threshold-driven mining over a raw time series. In interactive analysis of a fixed series, an equally natural contract is to preprocess once and repeatedly ask for the best K indexed patterns under different forgetting factors, length ranges, or values of K. Examples include repeated analyst exploration of a sensor archive, periodic monitoring dashboards, and ranking of market or environmental shapes at several recency horizons. In this setting, rebuilding candidate structures from the raw series for every query is avoidable if an index already materializes pattern loci and occurrence information.

The research gap has four layers. First, the *objective gap*: OPF-Miner models forgetting but is threshold based, whereas COPP-Miner is top-K but optimizes class contrast rather than temporal forgetting [6, 12]. Second, the *execution-model gap*: single-shot candidate generation is different from a build-once/query-many interface in which index construction is separated from query latency. Third, the *structural gap*: compact OP index edges contain loci with identical occurrence-start sets, but endpoint-based forgetting has not been used to turn such an edge into a reusable ranked stream. This step is non-trivial because the occurrence set is shared while the endpoint age changes with pattern depth; exploiting both facts requires separating a stream-specific occurrence term from a depth-specific decay term. Fourth, the *selection and numerical gap*: a convincing top-K contribution must be compared with capacity-K heaps and partial selection, not only with a full sort, and its floating-point score evaluation must avoid artificial underflow/overflow failure modes.

The direct product-form identity $\mathrm{fsup}_k(A, d) = M_k(A)e^{k(d-1)}$, where $M_k(A) = \sum_{s \in A} e^{-k(n-s)}$, is algebraically correct. It is nevertheless unsuitable as a machine-arithmetic contract: $M_k(A)$ can underflow while $e^{k(d-1)}$ overflows, even though a valid endpoint satisfies $s + d - 1 \leq n$ and hence every true endpoint weight lies in $(0, 1]$. We therefore anchor the factorization at the rightmost start of each stream. We retain binary64 because it is the native and efficient numeric type in the Java query path; the shift-safe form removes the identified NaN-producing intermediate

without imposing arbitrary-precision arithmetic on every interactive query, while an independent high-precision oracle checks semantic agreement on controlled cases.

The contributions are as follows.

1. We formalize a reusable indexed top-K forgetting query over weak-order pattern loci. The model defines weak-order equivalence, canonical dense-rank identity, predecessor–successor tie keys, the indexed locus universe, occurrence-start sets, maximal occurrence-equivalent streams, the eligible query set, and a deterministic total-order comparator.
2. We derive a *shift-safe occurrence factorization*. With $a = \max A$, one stream requires the shifted mass $R_k(A) = \sum_{s \in A} e^{-k(a-s)}$, after which every locus score uses only a nonpositive decay exponent. We prove equivalence to endpoint forgetting and give explicit binary64 safety bounds for the Java artifact.
3. We instantiate exact-real depth monotonicity as a comparator-consistent machine stream. Rounded-equal score plateaus are handled deterministically, and the numerical claim is deliberately limited to the declared Java binary64 comparator rather than arbitrary-precision ordering of adversarial near ties.
4. We instantiate ranked-stream pruning for the OPF setting: after retaining the best $\min(K, E)$ stream heads under the total comparator, a comparator-consistent multiway merge returns the exact top-K *under that machine contract*. Capacity-K selection, head pruning, and multiway merge are generic tools; the contribution is their use after the OPF-specific occurrence-equivalent factorization, not a new generic top-K theorem.
5. We provide a self-contained Java 21 artifact with five fair baselines/ablations, deterministic tests, an independent Python Decimal oracle, ten real benchmark instances drawn from nine public data products, controlled tie-rich synthetic generators, 30-run real-data benchmarks, a 40-run sensitivity campaign, 25-run synthetic benchmarks, and mixed repeated-query workloads. The largest real instance contains 500,000 observations. Results identify both favorable regimes and negative boundaries, including practical parity with the strongest heap baseline on the 500k Household instance, a B4-versus-B6 ablation, illustrative Top-K patterns, and a same-index demonstration that a forgetting threshold at the Kth score is not always output-equivalent to exact top-K.

The paper deliberately does not claim a new production OPST construction algorithm. The bounded weak-order prefix index exists so that the query-layer theorems and baselines can be reproduced without an external code dependency. Recent scalable OPST work shows that production indexing is a separate problem [10, 11]. None of the prior work reviewed below addresses the same combination of repeated global top-K ranking, endpoint exponential forgetting, weak ties, and a fixed occurrence-equivalent OP index.

The remainder of the paper is organized as follows. Section 2 positions the work relative to OP mining, indexed OP structures, generic top-K processing, and recency-aware stream mining. Section 3 formalizes the query model, and Section 4 gives a unified running example. Sections 5–6 develop the shift-safe factorization, stream

Table 1 Notation used throughout the paper. Symbols are grouped by indexed data, forgetting score, and query-work quantities.

Symbol	Meaning
$T = (t_1, \ldots, t_n)$, n	Input time series and its number of observations.
L, d	Maximum indexed pattern length and the depth/length of one pattern locus.
v, $A(v)$	Canonical weak-order pattern locus and its nonempty set of one-based occurrence starts.
$\mathrm{wnorm}(v)$, $\mathrm{PrefCode}(v)$	Dense-rank pattern identity and predecessor–successor tie-breaking code.
S_i, A_i, E, r_i, R	Occurrence-equivalent stream, its shared start set, number of eligible streams, eligible loci in stream i, and total eligible loci $R = \sum_i r_i$.
$a_i = \max A_i$	Rightmost occurrence start used to anchor stream i.
k, h	Exponential forgetting rate and half-life, related by $k = \ln 2/h$ when $h < \infty$.
$\mathrm{fsup}_k(v)$, $R_k(A)$	Endpoint forgetting support and shift-safe occurrence mass.
$Q = (k, K, \ell_{\min}, \ell_{\max})$	Reusable query: forgetting rate, requested result count, and eligible length interval.
$\prec_Q$, $K' = \min(K, E)$	Deterministic strict total query order and the maximum number of retained streams.
Z_{locus}, Z_{stream}	Occurrence incidences scanned by independent per-locus scoring and one-mass-per-stream scoring, respectively.
T_{plat}, R_K	Total head-plateau positions inspected before selection and eligible loci belonging to retained streams.

ordering, pruning, correctness, and complexity. Sections 7–8 describe baselines and experiments. Section 9 discusses scope and limitations, and Section 10 concludes.

2 Related work

2.1 Order-preserving matching and mining

Order-preserving matching formalizes equivalence between sequences that share relative order [1]. Order-preserving incomplete suffix trees and indexes established suffix-structure techniques for OP matching [2]. Subsequent mining methods expanded the objective space. OPP-Miner mines frequent order-preserving sequential patterns [3]; OPR-Miner mines order-preserving rules [4]; AOP-Miner allows approximate order-preserving patterns [5]; and later work addresses contrast, co-occurrence, outlier, and stability objectives [6–9]. The 2026 review by Zhou et al. [13] summarizes the broader OP matching landscape. EStream-OPF does not change OP equivalence itself; it assumes that an index exposes canonical pattern loci and their occurrence starts and contributes a weighted query layer over those objects.

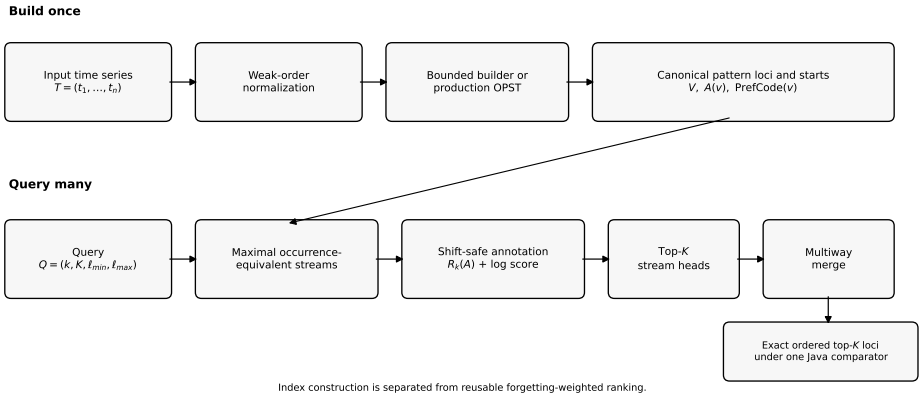


Fig. 1 Build-once/query-many architecture of Shift-Safe EStream-OPF. The build path materializes canonical weak-order loci and their occurrence starts using the reproducibility builder or a production OPST implementation. Each query then forms maximal occurrence-equivalent streams, evaluates one shift-safe mass per stream, retains the best stream heads, and performs a comparator-consistent multiway merge; index construction is therefore separated from the timed reusable query path.

2.2 Forgetting-aware OP mining

OPF-Miner is the closest predecessor in score semantics [12]. It assigns exponentially decaying weights to occurrence endpoints and searches for patterns satisfying a support threshold. Forgetting mechanisms also appear in incremental sequential mining, such as OIP-Miner [14]. These methods motivate recency-aware relevance, but they do not study repeated global top-K ranking over a materialized OP index. EStream-OPF therefore shares the endpoint-decay semantics while changing both the execution model and the selection objective.

2.3 Indexed OP structures

Scalable OP mining and indexed comparison work construct compact suffix structures and use predecessor/successor encodings to represent OP equivalence [10, 11]. We reuse this representational principle and do not claim the index structure itself as new. The interface needed here is narrower: each locus has a depth, a canonical pattern identity and tie key, and an occurrence-start set; a compact unary path can then expose maximal consecutive loci with equal occurrence sets. The 2026 indexing work is especially relevant because it demonstrates practical OPST construction and multi-million-length experiments, but its queries concern unweighted frequent/maximal/closed OP tasks rather than repeated forgetting-weighted top-K ranking [11].

2.4 Top-K pattern mining and ranked query processing

Threshold-free ranking is well established in pattern mining; TFP, for example, mines top-K frequent closed itemsets [15]. Database research has also developed general ranked-access principles. Fagin, Lotem, and Naor analyze threshold-style aggregation algorithms [16], and Ilyas et al. survey top-K query processing [17]. These foundations

Table 2 Scientific boundary. "Weak ties" means that equal values inside a window are retained in the canonical weak-order pattern rather than discarded. "Reusable OP index" means that repeated queries operate on a materialized OP-pattern index.

Method family	OP patterns	Temporal weighting	Top-K	Weak ties	Reusable OP index
OPF-Miner [12]	yes	exponential forgetting	no	method-specific	no
COPP-Miner [6]	yes	none	yes	method-specific	no
Indexed OP mining [11]	yes	none	no	yes	yes
Sliding-window statistics [19]	no	hard window	no	n/a	no
Stream top-K elements [20]	no	online counts	yes	n/a	no
Shift-Safe EStream-OPF	yes	exponential forgetting	yes	yes	yes

bound our novelty: a capacity-K heap, ranked-list heads, and multiway merge are not new. The Threshold Algorithm (TA) assumes sorted access to multiple score lists and a monotone aggregation function. Before our factorization, an OP edge is not a pre-existing set of independent attribute-score lists; after factorization, each occurrence-equivalent edge becomes a single ranked stream of alternative loci. The K-head rule is therefore not proposed as a replacement for TA, but as the simple pruning consequence that becomes usable after the OPF-specific stream transformation.

2.5 Recency-aware and top-K processing in data streams

A broader stream-mining literature addresses continuous frequency estimation, hard sliding windows, and frequent/top-K elements. Manku and Motwani give approximate frequency counting over streams [18]; Datar et al. formalize sliding-window statistics [19]; and Metwally et al. integrate frequent-element and top-K maintenance for streams [20]. These methods are important precedents for recency or continuous ranking, but their objects are stream elements or aggregate frequencies rather than OP pattern loci. They neither use endpoint exponential forgetting over pattern occurrences nor exploit occurrence-equivalent OP index paths. Thus the present gap is not merely "top-K on a stream"; it is exact reusable ranking over a fixed OP-pattern index with depth-dependent endpoint ages.

3 Preliminaries and formal query model

3.1 Weak-order normalization and equivalence

Let $x = (x_1, \ldots, x_m)$ be a finite real-valued window and let $\mathcal{V}(x)$ be the set of distinct values appearing in x. We use a tie-preserving dense-rank representation within the broader OP matching framework [1, 2].

Definition 1 (Weak-order normalization and equivalence) The dense-rank normalization of x is

$$\mathrm{wnorm}(x)_i = 1 + |\{v \in \mathcal{V}(x) : v < x_i\}|, \qquad 1 \le i \le m. \tag{1}$$

For equal-length windows x and y, define

$$x \sim_{\mathrm{wo}} y \quad \Longleftrightarrow \quad \mathrm{wnorm}(x) = \mathrm{wnorm}(y).$$

Equal values receive equal ranks, so ties are retained rather than discarded.

Lemma 1 (Weak-order equivalence and canonical dense ranks) *For a fixed length m, $\sim_{\mathrm{wo}}$ is an equivalence relation. Moreover,* $\mathrm{wnorm}(x)$ *uniquely records all pairwise relations $<$, $=$, and $>$ among positions in x.*

Proof Reflexivity, symmetry, and transitivity follow immediately because $\sim_{\mathrm{wo}}$ is equality of the deterministic vectors $\mathrm{wnorm}(\cdot)$. For any positions i, j, Eq. (1) assigns the same dense rank exactly when $x_i = x_j$, and assigns a smaller rank exactly when $x_i < x_j$. Hence the normalized vector determines, and is determined by, the window's weak ordering. Therefore it is a canonical representative of each equivalence class. □

For example, $(1,1,1,2,1,3,1)$ and $(10,10,10,20,10,30,10)$ have the same normalization $(1,1,1,2,1,3,1)$.

3.2 Predecessor–successor code

Pattern identity in the artifact is the dense-rank vector. For deterministic tie breaking we additionally use a predecessor–successor code in the style of OP suffix-tree research [2, 10, 11]. Let $p = (p_1, \ldots, p_m)$ be a dense-rank vector. For $i > 1$ define

$$J_i^- = \{j < i : p_j \le p_i\}, \qquad J_i^+ = \{j < i : p_j \ge p_i\}.$$

If $J_i^- \neq \emptyset$, let $v_i^- = \max\{p_j : j \in J_i^-\}$ and let $\pi_i^- = \max\{j < i : p_j = v_i^-\}$; otherwise set $\pi_i^- = \perp$. If $J_i^+ \neq \emptyset$, let $v_i^+ = \min\{p_j : j \in J_i^+\}$ and $\pi_i^+ = \max\{j < i : p_j = v_i^+\}$; otherwise set $\pi_i^+ = \perp$. We also set $(\pi_1^-, \pi_1^+) = (\perp, \perp)$. Then

$$\mathrm{LastCode}(p_{1:i}) = (\pi_i^-, \pi_i^+), \qquad \mathrm{PrefCode}(p) = \big(\mathrm{LastCode}(p_{1:1}), \ldots, \mathrm{LastCode}(p_{1:m})\big). \tag{2}$$

The implementation stores zero-based positions and encodes $\perp$ as -1. For $p = (1,2)$, $\mathrm{PrefCode}(p) = ((-1,-1),(0,-1))$. We emphasize that $\mathrm{wnorm}(p)$ establishes pattern identity; $\mathrm{PrefCode}(p)$ is a deterministic secondary ordering key, not a substitute definition of weak-order equivalence.

3.3 Indexed object model

Let a fixed time series $T = (t_1, \ldots, t_n)$ be indexed up to maximum depth L.

Definition 2 (Indexed weak-order object model) An index is

$$\mathcal{I} = (V, \mathrm{depth}, A, \mathrm{code}, \mathcal{S}),$$

where V is the finite set of canonical pattern loci; $\mathrm{depth}(v) \in \{1, \ldots, L\}$ is the pattern length; $A(v) \subseteq \{1, \ldots, n - \mathrm{depth}(v) + 1\}$ is the nonempty set of one-based occurrence starts; $\mathrm{code}(v) = \mathrm{PrefCode}(v)$ is the predecessor–successor tie key; and $\mathcal{S}$ is the stream partition defined next.

Definition 3 (Maximal occurrence-equivalent stream) A stream $S_i = (v_{i,1}, \ldots, v_{i,r_i})$ is a sequence of loci with strictly increasing depth and a shared nonempty occurrence-start set A_i:

$$A(v_{i,1}) = \cdots = A(v_{i,r_i}) = A_i.$$

It is maximal under the builder's unary-collapse rule: it cannot be extended toward its unique parent or unique child while preserving the same occurrence-start set.

The bounded reference builder creates one node for every observed weak-order pattern up to L. Every non-root node in this prefix tree has exactly one parent. A parent–child edge is collapsible only when the parent has one child and the two occurrence lists are identical.

Lemma 2 (Stream-partition invariant) *The bounded builder assigns every locus to exactly one maximal occurrence-equivalent stream; distinct streams are disjoint and their union is V.*

Proof Proceed by induction on locus depth. At depth one, every root-level locus has no parent and therefore starts exactly one stream. Assume every locus of depth below d has been assigned to exactly one stream. Let v have depth d. Because the structure is a prefix tree, v has a unique parent u. If the edge $u \to v$ is not collapsible, v starts a new stream. If it is collapsible, the induction hypothesis puts u in exactly one stream and the builder appends v to that stream. The unique-parent property prevents assignment through any second stream. Thus every depth-d locus is assigned exactly once. Induction proves coverage and pairwise disjointness. The builder continues each stream through every available unique child with the same occurrence list and stops precisely when that condition fails, so each resulting stream is maximal. □

Remark 1 (Index-interface scope) The query algorithm needs the partition invariant, not the bounded builder itself. A production OPST realization can expose the same abstraction by returning maximal indexed paths on which consecutive loci share one occurrence-start set. The present artifact uses the bounded builder only to make that interface self-contained and reproducible.

3.4 Forgetting support and reusable top-K query

For a locus v of depth d and any valid $s \in A(v)$, $1 \le s \le n - d + 1$, so its endpoint $s + d - 1$ does not exceed n. Following forgetting-aware OP mining [12], define

$$w_k(s, d) = e^{-k(n-(s+d-1))}, \qquad k \ge 0, \tag{3}$$

and

$$\mathrm{fsup}_k(v) = \sum_{s \in A(v)} e^{-k(n-(s+d-1))}. \tag{4}$$

For finite $k \ge 0$, every valid endpoint weight lies in $(0, 1]$; hence

$$0 < \mathrm{fsup}_k(v) \le |A(v)|. \tag{5}$$

Definition 4 (Reusable indexed top-K forgetting query) A query is $Q = (k, K, \ell_{\min}, \ell_{\max})$, where $k \ge 0$ is finite, $K \ge 1$, and $1 \le \ell_{\min} \le \ell_{\max} \le L$. The eligible universe is

$$\mathrm{Eligible}(Q) = \{v \in V : \ell_{\min} \le \mathrm{depth}(v) \le \ell_{\max}\}.$$

For any scalar score function $s_Q(v)$ that is never NaN, define the lexicographic query key

$$\kappa_Q(v) = \big(-s_Q(v), \mathrm{depth}(v), \mathrm{PrefCode}(v), \mathrm{wnorm}(v), \mathrm{id}(v)\big). \tag{6}$$

Candidate v is better than u, written $v \prec_Q u$, exactly when $\kappa_Q(v)$ is lexicographically smaller. Exact-real semantics use $s_Q(v) = \log \mathrm{fsup}_k(v)$. The Java implementation uses the shift-safe binary64 log score defined in Eq. (9). The query returns the best $\min(K, |\,\mathrm{Eligible}(Q)|)$ eligible loci.

Proposition 3 (Total query order) *For either exact-real log scores or the Java scores of Section 5.2, $\prec_Q$ is a strict total order on eligible loci.*

Proof Each component of Eq. (6) is totally ordered: scalar scores are ordered numerically (with $-\infty$ below every finite score in the Java contract), depth and unique ids are ordered integers, and the finite code and dense-rank vectors use lexicographic integer order. Lexicographic composition is transitive and total. The final unique locus id prevents two distinct loci from having identical complete keys, so the order is strict. □

The experiments set $\ell_{\min} = 2$ to exclude the trivial length-one pattern.

4 Running example

Consider

$$T = (1, 1, 1, 2, 1, 3, 1), \qquad n = 7,$$

with $L = 4$, $k = 0.2$, $K = 3$, and eligible lengths 2–4. The weak-order patterns $p_2 = (1, 2)$ and $p_3 = (1, 2, 1)$ both occur at starts $A = \{3, 5\}$. They therefore form one occurrence-equivalent stream, denoted S_A. Their predecessor–successor codes are

$$\mathrm{PrefCode}(p_2) = ((-1, -1), (0, -1)), \qquad \mathrm{PrefCode}(p_3) = ((-1, -1), (0, -1), (0, 0)).$$

Figure 2 shows the relevant bounded prefix-index structure. Heavy unary links connect nodes that share occurrence lists and therefore belong to one stream.

For S_A, $A = \{3, 5\}$ and $a = 5$. Its shift-safe mass is

$$\begin{aligned} R_{0.2}(A) &= e^{-0.2(5-3)} + e^{-0.2(5-5)} \\ &= e^{-0.4} + 1 = 1.6703200460. \end{aligned}$$

At depth $d = 2$, the rightmost occurrence ends at 6 and has age one, so

$$\mathrm{fsup}_{0.2}(A, 2) = e^{-0.2} R_{0.2}(A) = 1.3675423892.$$

At depth $d = 3$, the rightmost occurrence ends at 7 and has age zero, so

$$\mathrm{fsup}_{0.2}(A, 3) = R_{0.2}(A) = 1.6703200460.$$

Thus the head of S_A is $[1, 2, 1]$ (depth 3), and its next locus is $[1, 2]$ (depth 2).

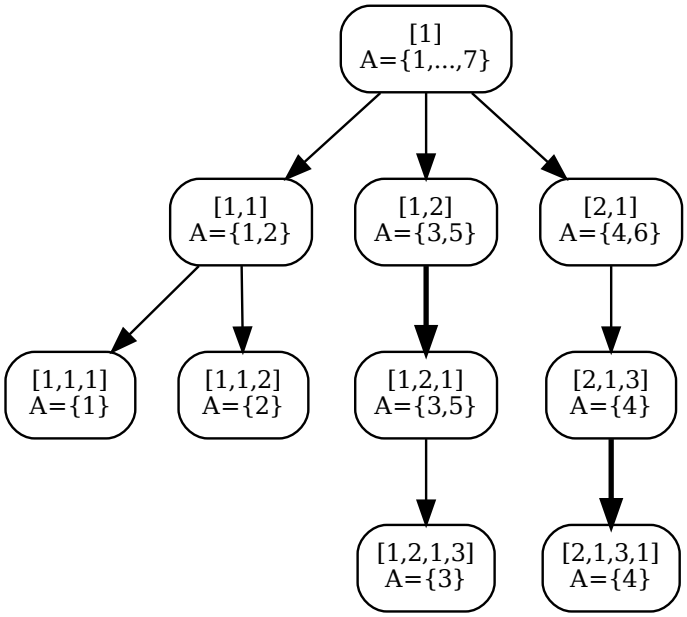


Fig. 2 Unified running example. Heavy unary links indicate occurrence-equivalent chains. The structure is the bounded reference index used for reproducibility, not a claim of a new production OPST constructor.

The separate one-locus stream S_B contains pattern $[2,1]$ with starts $B = \{4,6\}$. Here $b = 6$ and $R_{0.2}(B) = e^{-0.4} + 1 = 1.6703200460$; because its depth-2 rightmost endpoint is 7, its age is zero and $\mathrm{fsup}_{0.2}(B,2) = 1.6703200460$. A third relevant stream S_C contains $[2,1,3]$ and $[2,1,3,1]$ with start set $\{4\}$; its head $[2,1,3,1]$ has score 1.0.

Consequently, the three best *stream heads* are $[2,1]$ with score 1.6703200460, $[1,2,1]$ with the same score, and $[2,1,3,1]$ with score 1.0. The first two scores tie, so the total comparator places the shorter pattern $[2,1]$ first. The merge then emits $[1,2,1]$. Advancing only S_A reveals $[1,2]$ with score 1.3675423892, which outranks the still-unemitted S_C head. The global top three are therefore $[2,1]$, $[1,2,1]$, and $[1,2]$. This resolves the potential ambiguity between the two distinct depth-2 patterns/streams and illustrates why a non-head locus may enter the final answer only from a retained stream. At $k = 0$, loci within any one stream have equal support $|A|$, so shorter depth becomes the secondary ordering key.

5 Shift-safe occurrence factorization

5.1 Exact-real identity

Let S be a nonempty occurrence-equivalent stream with shared starts A. Since A is finite and nonempty, the rightmost start

$$a = \max A$$

exists. Define the shifted occurrence mass

$$R_k(A) = \sum_{s \in A} e^{-k(a-s)}. \tag{7}$$

Theorem 4 (Shift-safe factorization) *For every valid locus of depth d in the stream,*

$$\operatorname{fsup}_k(A, d) = e^{-k(n-(a+d-1))} R_k(A). \tag{8}$$

Proof For any $s \in A$,

$$n - (s + d - 1) = n - (a + d - 1) + (a - s).$$

Hence

$$e^{-k(n-(s+d-1))} = e^{-k(n-(a+d-1))} e^{-k(a-s)}.$$

The first factor is independent of s. Factoring it out of Eq. (4) and applying Eq. (7) gives Eq. (8). □

The exact logarithmic score is

$$\log \operatorname{fsup}_k(A, d) = \log R_k(A) - k(n - (a + d - 1)). \tag{9}$$

Baselines B1–B3 recompute the shifted mass independently for each locus; B4–B6 compute it once per occurrence-equivalent stream.

Corollary 5 (Exact-real stream order) *For $k > 0$, $\operatorname{fsup}_k(A, d)$ is strictly increasing with depth d within a nonempty occurrence-equivalent stream. For $k = 0$, every locus in the stream has support $|A|$ and shorter depth is first under the query comparator.*

Proof For $k > 0$, increasing depth by one reduces the nonnegative rightmost-endpoint age $n - (a + d - 1)$ by one and multiplies Eq. (8) by $e^k > 1$. For $k = 0$, every endpoint weight equals one. □

5.2 Numerical-safety contract

The product-form identity $M_k(A)e^{k(d-1)}$ is algebraically valid but can combine a mass rounded to zero with a positive exponential rounded to infinity. Such scale-separation failures are a standard reason to reformulate floating-point computations rather than rely on algebraic equivalence alone [21]. Equation (8) removes the positive exponential.

Quantitative binary64 failure regime.

For IEEE binary64, $\log(\texttt{Double.MAX_VALUE}) \approx 709.782712893384$, while the natural logarithm of the smallest positive subnormal value is approximately -744.440071921381. Consequently, the positive-depth factor in the direct product form overflows whenever $k(d-1) > 709.7827$, while a sufficiently old singleton mass can simultaneously underflow. A concrete valid instance is $n = 1000$, $A = \{200\}$, $d = 721$, and $k = 1$. The endpoint is $920 \le n$, so the exact forgetting support is $e^{-80} \approx 1.8048514 \times 10^{-35}$. In the direct product form, however, $M_1(A) = e^{-800}$ rounds to zero while e^{720} overflows, creating $0 \cdot \infty = \text{NaN}$. Under Eq. (8), $a = 200$, $R_1(A) = 1$, and the log score is simply -80. This example isolates a machine-arithmetic failure without changing the underlying exact-real score.

Relation to shifted log-sum-exp and accumulation error.

The numerical idea is related to the standard maximum-shift used to stabilize log-sum-exp calculations [21, 22]: both rewritings move exponential arguments into a nonpositive range. The present anchor is nevertheless structural rather than generic: $a = \max A$ is the rightmost occurrence time, and its use preserves a stream-specific mass plus a separate depth term, which is what makes repeated ranked-stream queries possible. For the measured datasets, $m = |A| \le n \le 500{,}000$. Conditional on the individually evaluated nonnegative exponential terms, standard floating-point summation gives the accumulation-only bound $|\widehat{R} - R|/R \le \gamma_{m-1}$, where $u = 2^{-53}$ and $\gamma_j = ju/(1-ju)$ [21]; hence $\gamma_{499999} < 5.56 \times 10^{-11}$. This is deliberately *not* claimed as a complete relative-error bound for fsup: it excludes the approximation error of each `StrictMath.exp` call, possible underflow of negligible old terms, and any collapse of distinct cross-stream exact scores to the same binary64 value. Those cases are handled by the declared machine comparator and the independent Decimal oracle rather than by an unsupported universal precision claim.

Theorem 6 (Shift-safe binary64 evaluation) *Assume $k \ge 0$ is finite and $A \ne \emptyset$ is a valid occurrence-start set represented by the Java artifact. Then: (i) every argument passed to `StrictMath.exp` while computing Eq. (7) is nonpositive or $-\infty$; (ii) the term for $s = a$ is exactly one; (iii) the computed shifted sum is positive and finite; (iv) the implementation never forms a product of a zero mass and a positive exponential; and (v) the Java log score is finite or $-\infty$, but never NaN or $+\infty$.*

Proof Because $a = \max A$, every distance $a - s$ is a nonnegative integer. For finite $k \ge 0$, the exact product $k(a-s)$ is nonnegative; if binary64 multiplication overflows, Java produces $+\infty$, whose negation is $-\infty$, and `StrictMath.exp` returns zero. No positive exponent is evaluated. For $s = a$, the integer distance is zero, so the exponent is exactly -0 and the contribution is exactly one. All other contributions lie in $[0, 1]$.

The artifact stores the series in a Java array and occurrence starts as `int`; therefore $|A| \le n \le 2^{31} - 1$. The exact shifted mass obeys $1 \le R_k(A) \le |A|$, and the binary64 accumulation of at most $2^{31}-1$ nonnegative terms bounded by one is many orders of magnitude below `Double.MAX_VALUE`; the computed sum is consequently finite and at least one. The rightmost endpoint is valid, so its age $n - (a + d - 1)$ is nonnegative. The implementation

ranks by $\log R_k(A) - k\,\text{age}$ and never constructs e^{+x}. If $k\,\text{age}$ overflows, the score becomes $-\infty$; otherwise it is finite. In neither case can it be NaN or $+\infty$. □

The Java comparator orders every finite log score ahead of $-\infty$ and then applies depth, PrefCode, dense-rank pattern, and unique-id tie keys. Thus even extreme underflow-to-$-\infty$ remains deterministic. The reported experiments are much smaller ($n \leq 500{,}000$), so the explicit artifact bound is conservative. We do not claim that naive summation is optimally accurate; Kahan/compensated summation could reduce rounding error, but it is unnecessary for eliminating the specific $0 \cdot \infty$ failure addressed here. High-precision Decimal arithmetic is retained as an independent oracle rather than placed on the latency-critical query path.

5.3 Comparator-consistent rounded plateaus

Even with safe evaluation, adjacent exact-real scores can round to the same binary64 logarithmic value. The machine comparator therefore ranks first by the computed log score, then by shorter depth and the deterministic keys of Eq. (6).

Lemma 7 (Linear construction of machine stream order) *Within one occurrence-equivalent stream, binary64 log scores produced by Eq. (9) are nondecreasing with depth under Java's fixed round-to-nearest binary64 operations. Hence an eligible stream can be placed in machine-comparator order in $O(r_i)$ time by processing rounded-score plateaus from highest to lowest and listing depths from shortest to longest inside each plateau.*

Proof The computed $\log R_k(A)$ is constant within the stream. As depth increases, the integer age decreases. Correctly rounded multiplication by a fixed nonnegative k is monotone in the nonnegative age, and correctly rounded subtraction from the same finite constant is monotone in the opposite direction; Section 5.2 excludes NaN. Rounding can therefore collapse adjacent values into an equal plateau but cannot reverse their order. Within a plateau, the comparator's depth key is ascending. □

Remark 2 (Scope of the machine-order claim) The lemma is an engineering property of the declared Java binary64 operations, not a certificate that every machine ordering equals arbitrary-precision real ordering for adversarial cross-stream near ties. The numerical stress suite covers k from 10^{-12} to 10^{100}, and an 80-digit Decimal oracle independently validates all reported small-query configurations. A certified interval or high-precision fallback for near-equal cross-stream scores is left to future work.

The proposed Java implementation uses a further optimization: before head selection, it evaluates the deepest score and scans backward only across an exactly equal machine-score plateau. Full comparator order is materialized only for the at most K retained streams.

6 Top-K stream selection and merge

Let h_i be the best eligible locus of stream S_i under the complete query comparator.

Algorithm 1: Shift-Safe EStream-OPF query (B6)

Input: Indexed streams $\mathcal{S}$; $Q = (k, K, \ell_{\min}, \ell_{\max})$
Output: Global top-K eligible loci under $\prec_Q$

$H \leftarrow$ empty capacity-K worst-first heap;
foreach *stream* S_i **do**
    identify its eligible interval; skip if empty;
    $a_i \leftarrow \max A_i$ and compute $\log R_k(A_i)$ once;
    find comparator-consistent head h_i from deepest score plus the rounded-equal plateau; offer (h_i, S_i) to H;
$Q_{pq} \leftarrow$ best-first priority queue;
foreach *retained stream in* H **do**
    construct comparator order for its eligible loci in linear time; push its head into Q_{pq};
out $\leftarrow [\,]$;
while $Q_{pq} \neq \emptyset$ *and* $|out| < K$ **do**
    pop globally best current locus x; append x to *out*;
    advance only x's stream; push the next locus if present;
return *out*;

Proposition 8 (K-head sufficiency) *Let E streams contain at least one eligible locus and let H_K be the best* $\min(K, E)$ *stream heads under the strict total order $\prec_Q$. No locus in a stream whose head is outside H_K can appear in the global top K under the same order.*

Proof If $E < K$, every stream is retained. Otherwise let $h_i \notin H_K$. Because $\prec_Q$ is a strict total order, score ties are already resolved by the remaining comparator keys; hence at least K distinct stream heads rank strictly ahead of h_i. By definition of a stream head, every locus $x \in S_i$ satisfies either $x = h_i$ or $h_i \prec_Q x$. Transitivity then places those K heads ahead of every $x \in S_i$, so no such x can have global rank at most K. □

This proposition is a ranked-stream pruning observation and is not claimed as a new generic top-K theorem. Its usefulness here follows from the OPF-specific transformation in Section 5, which produces one comparator-ordered stream from each occurrence-equivalent OP path.

B1 uses the same per-locus score loop as B2 but stores every eligible candidate and performs a full global sort. B3 uses the same score loop, then applies QuickSelect/partial selection before sorting only the selected prefix. Thus B1–B3 differ in selection strategy, not score semantics.

Theorem 9 (Merge correctness under the Java comparator) *Algorithm 1 returns the globally best* $\min(K, |\,\text{Eligible}(Q)|)$ *eligible loci, in order, under the declared machine comparator $\prec_Q$.*

Algorithm 2: AllLoci-TopKHeap baseline (B2)

Input: All indexed loci V; $Q = (k, K, \ell_{\min}, \ell_{\max})$
Output: Global top-K eligible loci under $\prec_Q$

```
H ← empty capacity-K worst-first heap;
foreach v ∈ V with ℓ_min ≤ depth(v) ≤ ℓ_max do
    evaluate the same shift-safe log score independently for A(v);
    offer v to H; if |H| > K, remove the worst candidate;
copy H to a list and sort it by ≺_Q;
return the ordered list;
```

Proof By K-head sufficiency, all discarded streams are irrelevant. For the base case, before the first pop the priority queue contains exactly the best not-yet-emitted locus of every retained stream, so its minimum key is the globally best remaining eligible locus. For the induction step, assume the first $j < K$ outputs are globally correct. After emitting output j, the algorithm advances only that output's stream and replaces its head by that stream's next best unseen locus; every other stream head is unchanged. Therefore the queue again contains the best unseen locus from every active retained stream, and its next pop is the globally best remaining locus. Induction proves correctness for all emitted results. □

6.1 Complexity

Let E be the number of eligible streams, r_i the number of eligible loci in stream i, and $R = \sum_i r_i$ the total eligible loci. Because all r_i loci in stream i share A_i, distinguish

$$Z_{\text{locus}} = \sum_{i=1}^{E} r_i |A_i|, \qquad Z_{\text{stream}} = \sum_{i=1}^{E} |A_i|.$$

The first quantity is the occurrence incidence scanned by independent per-locus scoring (B1–B3); the second is scanned when one shifted mass is computed per stream (B4–B6), and $Z_{\text{stream}} \le Z_{\text{locus}}$. Let $K' = \min(K, E)$, let R_K be the total eligible loci in retained streams, and let T_{plat} be the total size of rounded-equal head plateaus inspected before head selection.

B2 has time

$$O\big(Z_{\text{locus}} + R \log K\big), \tag{10}$$

where the first term is score aggregation and the second is capacity-K heap maintenance. A portable B6 realization that explicitly orders every stream has worst-case time

$$O\big(Z_{\text{stream}} + R + E \log K' + K \log K'\big). \tag{11}$$

The final term is at most K priority-queue pops/pushes, each costing $O(\log K')$. The implemented lazy-head version usually avoids scoring all R loci before selection and performs work closer to

$$O\big(Z_{\text{stream}} + T_{\text{plat}} + E \log K' + R_K + K \log K'\big). \tag{12}$$

Table 3 Asymptotic query-work summary for B1–B6 on an already-built index. The B3 worst case reflects the deterministic last-element pivot used by the supplied QuickSelect implementation; its expected non-adversarial selection cost is linear. B5/B6 expressions follow the implemented lazy stream cursors.

ID	Time bound / practical work expression	Dominant distinction
B1	$O(Z_{\text{locus}} + R \log R)$	Scores every eligible locus and full-sorts all R candidates.
B2	$O(Z_{\text{locus}} + R \log K)$	Scores every locus; capacity-K heap is the principal competitive baseline.
B3	$O(Z_{\text{locus}} + R^2 + K \log K)$ worst case; expected $O(Z_{\text{locus}} + R + K \log K)$ for non-adversarial partitions	Same per-locus scores, then QuickSelect and a sort of the selected prefix.
B4	$O(Z_{\text{stream}} + R \log K)$	Reuses one shifted mass per stream but still offers every eligible locus to a heap.
B5	$O(Z_{\text{stream}} + T_{\text{plat}} + E \log E + K \log E)$ plus lazy cursor advances	Retains all eligible stream heads; removes K-head filtering.
B6	$O(Z_{\text{stream}} + T_{\text{plat}} + E \log K' + R_K + K \log K')$ practical lazy-head work	Proposed factorization + capacity-K head filtering + merge of retained streams.

Equations (10)–(12) show the target regime directly: B6 can replace repeated occurrence aggregation Z_{locus} by Z_{stream} and avoid most locus-level heap offers when $K \ll E$, but its stream bookkeeping can lose when compression is shallow or K/E is large.

The query selector and merge queue each contain at most K' stream states. Excluding output and the small retained-stream order arrays, extra query space is $O(K'L)$ in the bounded implementation and can approach $O(K')$ with direct interval iterators. We report structural counters rather than byte-level memory because the benchmark campaign did not retain JFR/RSS allocation traces; no memory-efficiency claim is made.

The bounded builder is intentionally not assigned production-OPST complexity. Its weak-order normalization sorts each window, giving the straightforward bound

$$O\left(\sum_{d=1}^{L}(n-d+1)d\log d\right)$$

plus hash-map and occurrence-list costs. A "raw-rebuild" execution that reconstructs this bounded index for every query would add this construction cost to B2 (or another selector) and violates the build-once query contract. We therefore do not present it as a competitive indexed-query baseline; instead, Section 8.7 reports measured construction separately and quantifies its amortization. Likewise, a production OPST constructor is an alternative index-building layer, not a substitute top-K selector, and is reserved for future end-to-end integration.

Table 4 Implemented Java 21 baselines and ablations.

ID	Query method	Main role
B1	Score every eligible locus and globally sort all loci.	Full-sort reference/ablation
B2	Score every eligible locus and retain a capacity-K heap.	Competitive top-K baseline
B3	Score all loci, partial-select the top prefix, then sort the selected prefix.	Checks whether gains are only from avoiding full sort
B4	Compute one shifted mass per stream but submit every eligible locus to a capacity-K heap.	Isolates factorization benefit
B5	Compute every stream head and initialize the merge PQ with all eligible streams.	Isolates K-head pruning
B6	Shift-safe mass + capacity-K head selection + lazy merge of retained streams.	Proposed method

7 Baselines and implementation

Table 4 lists all compared indexed-query algorithms. B2 is the principal competitive baseline because it uses a capacity-K heap rather than a full sort; B3 checks partial selection; B4 isolates stream-level factorization; and B5 removes the K-head filter while keeping stream merging. B1 remains a full-sort reference. All six algorithms consume the same already-built index, use the same shift-safe score semantics, and end with the same total comparator. A raw-series rebuild and production OPST construction are discussed separately because they change the index-construction contract rather than only the query selector.

All timed algorithms use the same Java data structures, the same bounded index, the same weak-order identity, and the same final comparator. The numerical utility class uses `StrictMath`. There is no epsilon-based ranking or pruning. The Python Decimal component is used only as an independent semantic oracle, not for performance measurement.

7.1 Protocol boundary to OPF-Miner

OPF-Miner [12] is the closest predecessor in forgetting semantics and is therefore essential for conceptual comparison, but a raw wall-clock ratio would not be an output-equivalent baseline for the query studied here. OPF-Miner starts from the raw series, generates/fuses candidates, and returns every pattern that satisfies a minimum forgetting-support threshold. EStream-OPF assumes a materialized OP index and

Table 5 Protocol comparison with OPF-Miner. The two methods share endpoint exponential forgetting but solve different selection and execution contracts; consequently no fabricated cross-objective runtime ratio is reported.

Aspect	OPF-Miner [12]	Shift-Safe EStream-OPF
Input contract	Raw time series; mining performs candidate discovery/fusion.	Already-built weak-order index exposing loci, starts, and occurrence-equivalent streams.
Temporal score	Endpoint exponential forgetting support.	Same endpoint exponential forgetting semantics, evaluated by the shift-safe log formulation.
Selection objective	Threshold-driven: enumerate patterns meeting a minimum forgetting-support condition.	Exact global top-K ranking over an explicit eligible length interval.
Output cardinality	Data- and threshold-dependent.	Fixed at $\min(K, \lvert\,\text{Eligible}(Q)\rvert)$.
Construction cost	Intrinsic to the single-shot mining run.	Separated from query latency and amortized in the repeated-query experiment.
Directly comparable evidence	Forgetting-score semantics and returned-order correctness on controlled cases.	Java B1–B6 share one indexed query contract; Decimal oracle checks score/order semantics.
Runtime interpretation	Measures threshold mining from raw input.	Measures repeated ranked selection from a materialized index.

returns exactly the best K eligible indexed loci. Matching a threshold post hoc to force OPF-Miner to emit K results would make its candidate-generation work and output cardinality depend on an unknown, data-specific threshold and would still include a different construction contract. Table 5 therefore separates the semantics that can be compared directly from the execution costs that cannot be interpreted fairly as one speedup number.

Accordingly, B2 is the strongest *output-equivalent* baseline for the proposed indexed top-K selector, while OPF-Miner is the closest *problem-semantic* predecessor. The same index already shows that the two selection contracts are not interchangeable even when they share scores. On Nile at the core setting, 747 loci are eligible and the exact top-50 cutoff score is attained by 55 loci; a threshold miner that keeps every pattern with $\text{fsup}_k \geq \text{fsup}_{(K)}$ therefore emits five extra tied patterns. On weekly CO_2 the same construction returns exactly 50 loci, so a cutoff at the Kth score happens to match top-K cardinality. Table 6 records both cases. A future end-to-end study can integrate the same production OPST layer with both a threshold mining interface and the present ranked-query interface, reporting construction, candidate generation, and selection separately. This distinction prevents an apparent speedup from being created merely by timing non-equivalent tasks.

Table 6 Same-index illustration that a forgetting threshold at the Kth top-K score is not always output-equivalent to exact top-K. Counts are from the Java artifact at $L = 12$, $K = 50$, $h = n/4$.

Dataset	Eligible loci	Top-K size	Loci with $\text{fsup}_k \geq \text{fsup}_{(K)}$
Nile annual	747	50	55
CO_2 weekly	11,245	50	50

8 Experimental evaluation

8.1 Research questions

The evaluation is organized around six questions: **RQ1** correctness and numerical safety; **RQ2** reduction in occurrence aggregation from the shift-safe factorization; **RQ3** work reduction from K-head pruning relative to an all-stream priority queue; **RQ4** competitive latency against full-sort, capacity-K heap, partial-selection, and factorization-only baselines; **RQ5** sensitivity to real-series size, compression, L, K, forgetting horizon, and ties; and **RQ6** amortized benefit under build-once/query-many workloads.

8.2 Real-world and synthetic datasets

The artifact contains ten real-series benchmark instances plus a deterministic synthetic family. The four compact reference series are exported through the statsmodels dataset interface [23]. They represent solar activity, ocean/climate, atmospheric measurements, hydrology, financial markets, urban air-quality sensing, and household energy use. The two Household rows are nested prefixes of the same UCI source series; hence the ten benchmark instances are drawn from nine public source data products. Dataset provenance and preprocessing are summarized in Table 7. The source package also records the original URLs and preprocessing scripts. For CO_2, the exporter interpolates missing weekly values and edge-fills the ends. For UCI Air Quality, documented missing-coded readings are removed before the benzene series C6H6(GT) is used. For UCI Household Power, the experiment takes the first 100,000 and 500,000 valid Global Active Power observations. The largest tested real instance therefore has 500,000 values; this is a stress test for the bounded reproducibility builder, not a claim to match the multi-million-scale construction performance of production OPST implementations [10, 11].

Controlled synthetic series combine two sinusoidal components, a bounded autoregressive component with random-walk variation, additive noise, and optional quantization. Seeds and quantization steps are fixed in the Java source; the principal scaling experiment uses quantization step 0.5 to generate frequent weak-order ties.

8.3 Implementation, machine configuration, and protocol

All timed algorithms are implemented in Java 21 and use the same weak-order index, data structures, score utility, and total-order comparator. The benchmark container

Table 7 Real-world benchmark instances. n, minimum and maximum values are from the exact input snapshots used in `publication_matrix.csv`.

Instance	Domain	n	Value range	Public source	Preprocessing / selected signal
Nile annual	Hydrology	100	456.00–1,370.00	statsmodels [24]	numeric annual series
Sunspots yearly	Solar activity	309	0–190.2	statsmodels [25]	numeric annual series
El Niño monthly	Climate/ocean	732	18.95–29.24	statsmodels [26]	numeric monthly SST series
CO_2 weekly	Atmosphere	2,284	313–373.9	statsmodels [27]	weekly series; missing weeks interpolated and edge-filled
S&P 500	Finance	2,512	2,085.18–7,798.99	FRED [28]	daily close snapshot; numeric observations
UCI Air Quality	Air-quality sensor	8,991	0.1–63.7	UCI [29]	C6H6(GT); missing-coded rows removed
NASDAQ Composite	Finance	14,015	54.87–27,093.90	FRED [30]	daily close snapshot; numeric observations
SILSO daily	Solar activity	72,967	0–528	WDC-SILSO [31]	daily total sunspot number; missing code removed
UCI Household 100k	Household energy	100,000	0.194–9.41	UCI [32]	first 100,000 valid Global_active_power observations
UCI Household 500k	Household energy	500,000	0.082–10.67	UCI [32]	first 500,000 valid Global_active_power observations

exposed four processors on an AMD EPYC 9V74 host running Linux 6.18.35 on x86-64 and OpenJDK 21.0.11 (Debian). Timed runs used `-Xms256m -Xmx2g -XX:+UseG1GC`. `System.nanoTime()` surrounds only the query call; file I/O and index construction are excluded from query latency and reported separately. The main real-data campaign uses 30 warm-up iterations followed by 30 measured repetitions for each algorithm/query and reports median latency with Q1 and Q3 in the raw CSV. The independent length-sensitivity sweep also uses 30/30 runs. The unified K/half-life sensitivity uses 50 warmups and 40 measurements, the synthetic scaling sweep uses 20 warmups and 25 measurements, and the repeated-query experiment uses 40 query warmups followed by seven measured batches. Unless otherwise stated, $L = 12$, $K = 50$, $\ell_{\min} = 2$, and forgetting half-life $h = n/4$, so $k = \ln(2)/h$.

The primary timing summary uses medians and interquartile ranges because JVM microbenchmarks are sensitive to scheduling, JIT, and garbage-collection noise. Overlapping IQRs on Household 500k are treated as practical parity. Individual repetitions were not retained in the supplied publication snapshot, so we do not invent run-level significance tests; Section 8.5 instead adds a transparent paired analysis across dataset medians and a sensitivity analysis that removes the nested Household 100k prefix. All latency ratios are interpreted only within the same timing campaign; the independent length and unified-sensitivity sweeps are not cross-compared against publication-matrix medians. A leftover file `core_latency_30runs.csv` is retained only as a historical snapshot of an earlier four-series campaign; it is *not* the source of Tables 9–10 and must not be mixed with `publication_matrix.csv`. The main campaign runs B1–B6 in a fixed algorithm order rather than a shuffled order; later algorithms may therefore receive additional JIT/GC sharing. Peak JVM/RSS memory was not captured, so the paper makes no byte-level memory-performance claim; controlled JMH/JFR profiling remains future validation.

8.4 RQ1: correctness and numerical stress

The Java test campaign verifies 1,080 randomized query configurations; B1–B6 return identical ordered top-K locus lists in all tested cases. A numerical stress suite performs 114 machine-score evaluations for forgetting rates from 10^{-12} through 10^{100} and observes no NaN from the shifted formulation. A dedicated regression case reproduces the old product-form failure $0 \cdot \infty = \text{NaN}$ while the shift-safe log score remains ordered (Table 8). Independently, an 80-digit Python Decimal brute-force oracle validates 75 small top-K configurations, including $k = 0$, $k = 1/n$, tied random series, and the running example. These checks validate the implementation against both an in-language

Table 8 Constructed binary64 regression for the unsafe product form $M_k(A)e^{k(d-1)}$ versus the shift-safe log score. The occurrence $A = \{200\}$, $d = 721$ is a valid window of a length-1000 series.

Form	$M_k(A)$	$e^{k(d-1)}$	Result
Product $M_k(A)e^{k(d-1)}$	0 (underflow)	$+\infty$ (overflow)	NaN
Shift-safe $\widehat{\ell}$	—	—	-80 (finite); fsup $= e^{-80}$

Table 9 Core real-data query latency (median ms, 30 measured runs after 30 warmups), $L = 12$, $K = 50$, $h = n/4$. Bold is the fastest median in each row.

Dataset	B1 full sort	B2 heap	B3 partial	B4 fact.-heap	B5 all-stream PQ	B6 EStream	B2/B6
Sunspots yearly	0.240	0.103	0.133	0.094	0.099	**0.093**	1.10×
El Niño monthly	0.673	0.351	0.390	**0.324**	0.399	0.450	0.78×
CO_2 weekly	2.731	1.347	1.576	0.808	0.841	**0.782**	1.72×
Nile annual	0.093	0.065	0.038	0.034	**0.031**	0.040	1.61×

cross-algorithm invariant and an independent high-precision oracle. The core real-data campaign uses $k = \ln 2/(n/4)$ and $d \le 12$, so $k(d-1) \ll 709$; the reported speedups are therefore *not* caused by avoiding NaN on those series. All six algorithms already consume the same shift-safe scores. The numerical contract is a robustness property of the query layer, isolated experimentally by the constructed overflow example rather than by the ten-instance latency matrix.

8.5 RQ2–RQ4: structural work and competitive real-data latency

Table 9 first reports B1–B6 on the four original real series. The strongest general all-loci baseline is B2, which evaluates every eligible locus but retains only a capacity-K heap; B3 provides a partial-selection alternative. B4 isolates the benefit of computing one shift-safe mass per occurrence-equivalent stream, while B5 isolates the cost of merging all stream heads without K-head pruning.

Across all ten real instances, B6 has a lower median than B2 on **8 of 10** rows. The largest gains occur on CO_2 (1.72×), SILSO daily sunspots (1.69×), UCI Household 100k (1.64×), Nile (1.61×), and NASDAQ Composite (1.43×). These are relative median speedups versus B2, not claims of universal dominance. B6 is the absolute fastest of B1–B6 on four rows (Sunspots, CO_2, S&P 500, and UCI Air Quality); B4 is fastest on El Niño, NASDAQ, SILSO, and Household 100k; B5 is fastest on Nile; and B2 is fastest on Household 500k. Table 11 isolates the two query-layer ingredients: sharing one shifted mass (B4) versus adding K-head pruning (B6). B4/B6 lies in $[0.95, 1.06]$ on seven of ten rows, so factorization—not head filtering—is the dominant, portable saving. Head pruning helps when compression is high and K/E is small (CO_2, S&P 500, Air Quality) and can lose when the index is shallow or tiny (El Niño, Household 500k, Nile).

The structural counters explain why median latency does not follow n alone. B6 reduces occurrence-list annotation by 14.7–58.3% relative to B2 and reduces machine

Table 10 Expanded real-world campaign from the uploaded `publication_matrix.csv` (median ms, 30 measured runs after 30 warmups). "Best" is the fastest B1–B6 median.

Dataset	n	Loci	Streams	Loci/stream	B2	B6	B2/B6	Best
Nile annual	100	748	150	4.99	0.065	0.040	1.61×	B5
Sunspots yearly	309	1,507	491	3.07	0.103	0.093	1.10×	B6
El Niño monthly	732	2,416	1,105	2.19	0.351	0.450	0.78×	B4
CO_2 weekly	2,284	11,246	3,212	3.50	1.347	0.782	1.72×	B6
S&P 500	2,512	13,495	3,839	3.52	0.783	0.704	1.11×	B6
UCI Air Quality	8,991	40,772	12,953	3.15	4.133	3.738	1.11×	B6
NASDAQ Composite	14,015	56,962	19,989	2.85	7.411	5.169	1.43×	B4
SILSO daily	72,967	287,789	93,489	3.08	61.961	36.660	1.69×	B4
UCI Household 100k	100,000	405,574	143,018	2.84	157.423	95.919	1.64×	B4
UCI Household 500k	500,000	1,343,832	575,349	2.34	451.939	452.753	1.00×	B2

Table 11 Ablation of stream-mass reuse (B4) versus B6 K-head pruning on the publication matrix (median ms). B4/B6 < 1 means factorization without head filtering is faster.

Dataset	Loci/stream	K/E	B4	B6	B4/B6
Nile annual	4.99	0.336	0.034	0.040	0.84×
Sunspots yearly	3.07	0.102	0.094	0.093	1.01×
El Niño monthly	2.19	0.045	**0.324**	0.450	0.72×
CO_2 weekly	3.50	0.016	0.808	**0.782**	1.03×
S&P 500	3.52	0.013	0.746	**0.704**	1.06×
UCI Air Quality	3.15	0.004	3.796	**3.738**	1.02×
NASDAQ Composite	2.85	0.003	**4.887**	5.169	0.95×
SILSO daily	3.08	0.001	**35.242**	36.660	0.96×
UCI Household 100k	2.84	<0.001	**93.369**	95.919	0.97×
UCI Household 500k	2.34	<0.001	624.675	452.753	1.38×

score evaluations by 28.9–65.2% on the real rows, while head pruning discards 66.4–99.99% of eligible streams. Yet those reductions do not guarantee faster wall-clock time because annotation, cursor construction, and priority-queue operations have different constants. The 500k Household instance is the clearest boundary: its compression is only 2.34 loci/stream and annotation falls by just 14.7%; B2 has median 451.939 ms and B6 452.753 ms with strongly overlapping IQRs, so the result is reported as parity rather than as a win or loss of practical importance. On that row B4 is slower still (624.675 ms), showing that stream bookkeeping without K-head filtering can lose when R is large and compression is shallow.

Table 13 lists the five highest-scoring eligible loci on Nile and weekly CO_2 under the core query. Both rankings are dominated by short, frequent monotone motifs whose rightmost occurrence is near the series end, which is the intended recency bias: Nile ranks the length-2 descent $[2, 1]$ and ascent $[1, 2]$ first, whereas CO_2 ranks the globally common ascent $[1, 2]$ and then longer strictly increasing prefixes. The table is a semantic sanity check, not a domain-scientific finding.

8.6 Cross-instance statistical interpretation

The timing files distributed with this manuscript retain the median and interquartile range of each 30-run algorithm/query campaign, not every individual repetition. We therefore do not manufacture run-level p-values after the fact. As a complementary

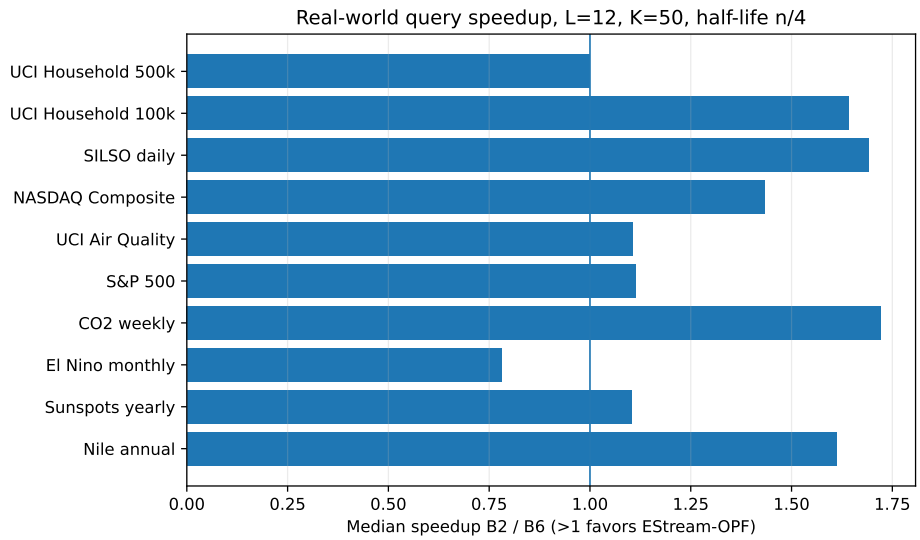


Fig. 3 Median B2/B6 speedup on the ten real benchmark instances. The vertical reference at 1.0 denotes parity; values to the right favor EStream-OPF.

Table 12 Large real-series subset. Entries are median [Q1,Q3] query milliseconds; annotation reduction compares B6 with B2.

Dataset	n	B2 ms [IQR]	B6 ms [IQR]	B2/B6	Annotation reduction (%)
UCI Air Quality	8,991	4.133 [3.974,4.464]	3.738 [3.687,4.154]	1.11×	29.5
NASDAQ Composite	14,015	7.411 [6.991,7.756]	5.169 [4.770,5.610]	1.43×	25.0
SILSO daily	72,967	61.961 [61.377,62.527]	36.660 [35.836,37.126]	1.69×	24.9
UCI Household 100k	100,000	157.423 [155.584,159.935]	95.919 [94.160,96.707]	1.64×	24.7
UCI Household 500k	500,000	451.939 [314.631,511.284]	452.753 [281.339,540.648]	1.00×	14.7

dataset-level analysis, we pair the B2 and B6 *instance medians* and apply an exact Wilcoxon signed-rank test to log(B2/B6). Across all ten benchmark instances, the geometric-mean ratio is 1.278, the matched-pairs rank-biserial effect is 0.782, and the two-sided exact Wilcoxon value is $p = 0.0273$ ($W_- = 6$, $W_+ = 49$). Because the 100k and 500k Household rows are nested prefixes and are therefore not independent source products, we also repeat the calculation using only the 500k Household prefix: across nine distinct source products the geometric-mean ratio is 1.243, rank-biserial effect is 0.733, and the exact two-sided result is $p = 0.0547$ ($W_- = 6$, $W_+ = 39$). These figures are treated as sensitivity evidence, not as a license for a universal speedup claim. The primary evidence remains the per-instance median, IQR, structural counters, and explicit negative boundary on Household 500k. The calculation is reproduced by `scripts/cross_instance_stats.py`.

8.7 RQ5: sensitivity to compression, K, forgetting horizon, and synthetic size

An independent CO_2 30-run length sweep varies the maximum indexed depth. Because it is a separate timing campaign, its $L = 12$ row is *not* substituted by the main

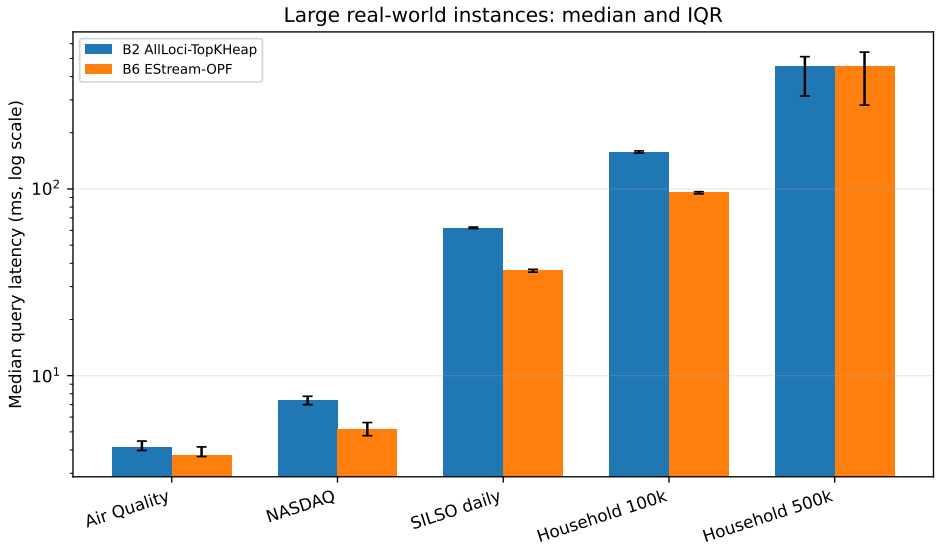


Fig. 4 B2 and B6 on the five largest real instances, with interquartile-range error bars. A log latency axis allows the 9k–500k settings to be compared in one panel.

Table 13 Highest-scoring eligible loci on two real series at $L = 12$, $K = 50$, $h = n/4$. Scores are fsup_k; $a_{\max}$ is the rightmost one-based occurrence start. Produced by `IllustrativeDump` in the Java artifact.

Series	Rank	Depth	$\lvert A\rvert$	$a_{\max}$	fsup_k	Pattern
Nile ($n = 100$)	1	2	51	98	17.735	$[2, 1]$
	2	2	47	99	16.411	$[1, 2]$
	3	3	20	98	7.140	$[2, 1, 3]$
	4	3	17	97	6.298	$[3, 2, 1]$
	5	3	17	96	5.828	$[2, 3, 1]$
CO_2 ($n = 2{,}284$)	1	2	1,177	2,283	400.198	$[1, 2]$
	2	2	935	2,272	319.230	$[2, 1]$
	3	3	702	2,282	236.310	$[1, 2, 3]$
	4	3	490	2,268	168.941	$[3, 2, 1]$
	5	4	436	2,281	145.042	$[1, 2, 3, 4]$

campaign's $L = 12$ measurement; this avoids mixing snapshots from different JVM warmup histories. In that sweep, compression rises from 1.82 loci/stream at $L = 8$ to 8.20 at $L = 20$. B2 is faster at $L = 8$ (B2/B6=0.71), whereas B6 is faster at $L = 12, 16, 20$ with ratios 1.36, 1.61, and 1.96. The crossover supports the structural hypothesis but also shows that compression alone is not a complete cost model.

A unified 40-run sensitivity campaign at $L = 16$ measures the $K = 50, h = n/4$ anchor once and reuses that same observation in the K and forgetting sweeps. For $K \in \{10, 50, 200, 500\}$, B2/B6 ranges from 1.18 to 1.85; the smallest advantage appears at $K = 500$, where retaining many stream heads reduces pruning leverage. Across $k = 0$ and finite half-lives $h \in \{n, n/2, n/4, n/16\}$, B2/B6 ranges from 1.42 to 1.97.

Table 14 Paired cross-instance sensitivity based on B2/B6 median latency ratios. The nine-product row removes the nested Household 100k prefix.

Analysis unit	N	Geom. mean B2/B6	Rank-biserial	Exact p
Benchmark instances	10	1.278	0.782	0.0273
Distinct source products	9	1.243	0.733	0.0547

Table 15 Independent CO_2 maximum-length sensitivity (30 warmups, 30 measured runs).

L	Loci	Streams	Loci/stream	B2/B6
8	3,748	2,065	1.82	0.71×
12	11,246	3,212	3.50	1.36×
16	20,083	3,524	5.70	1.61×
20	29,084	3,545	8.20	1.96×

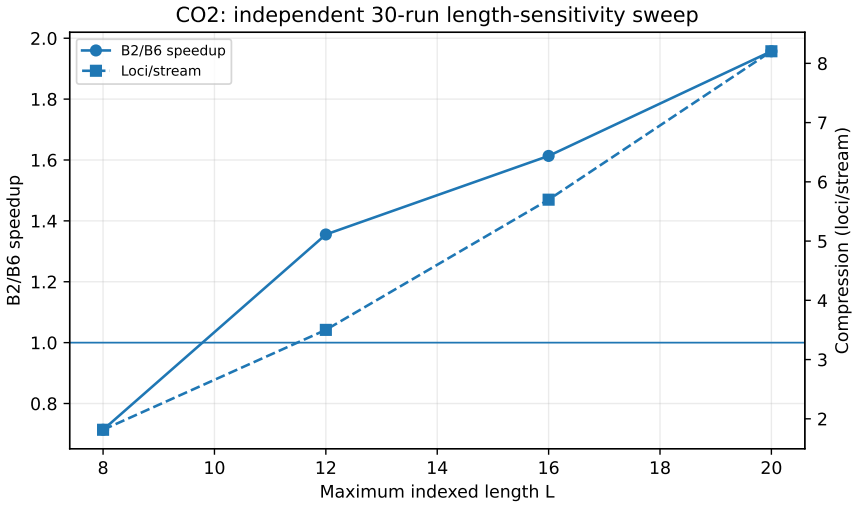


Fig. 5 CO_2 length sensitivity. The speedup curve and loci-per-stream compression are shown together; the experiment is a separate 30-run sweep from the main publication matrix.

Thus the method is not tied to a single recency horizon, although JVM-level latency remains workload dependent.

On controlled tie-rich synthetic data (quantization step 0.5), B6 is faster than B2 at every tested final size: B2/B6 is 1.35× at $n = 1,000$, 1.64× at 5,000, 1.07× at 10,000, 2.02× at 25,000, and 2.34× at 50,000. The non-monotone ratios caution against equating series length or compression with speedup. A separate tie-sensitivity

Table 16 Unified CO_2 sensitivity at $L = 16$ (50 warmups, 40 measured runs); medians in ms.

K	B2	B6	B2/B6	Half-life h	B2	B6	B2/B6
10	1.068	0.577	1.85×	∞	0.693	0.447	1.55×
50	1.317	0.926	1.42×	n	0.994	0.505	1.97×
200	1.095	0.648	1.69×	$n/2$	1.136	0.597	1.90×
500	1.234	1.045	1.18×	$n/4$	1.317	0.926	1.42×
				$n/16$	1.084	0.553	1.96×

Table 17 Tie-rich synthetic scaling ($L = 12$, $K = 50$, $h = n/4$), 25 measured runs after 20 warmups.

n	Loci	Streams	B2	B6	B2/B6
1,000	7,035	1,515	0.700	0.520	1.35×
5,000	30,642	7,580	1.900	1.161	1.64×
10,000	57,783	15,025	3.187	2.968	1.07×
25,000	131,533	37,522	26.404	13.055	2.02×
50,000	244,502	74,777	68.179	29.177	2.34×

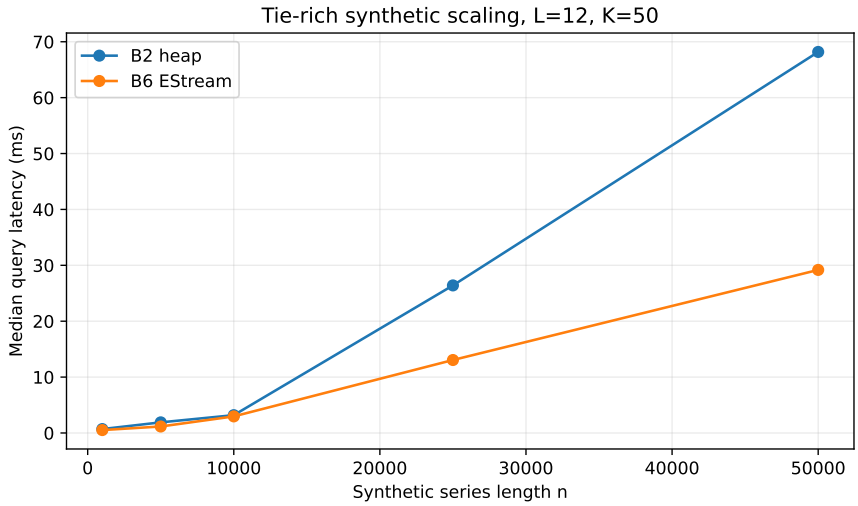


Fig. 6 Tie-rich synthetic scaling for B2 and B6.

test ranges from effectively continuous data to approximately 88.5% tied windows and preserves the weak-order correctness invariant throughout.

8.8 RQ6: build-once/query-many reuse

Index construction is explicitly separated from query latency. On CO_2 at $L = 16$, the median bounded-index build cost is 89.696 ms. The repeated-query batches reuse

Table 18 Repeated-query CO_2 workload at $L = 16$; build-amortized ms/query, median over seven measured batches.

Q	B2	B6	B2/B6
1	90.947	90.254	1.01×
10	10.061	9.634	1.04×
100	1.775	1.506	1.18×
1,000	1.019	0.732	1.39×

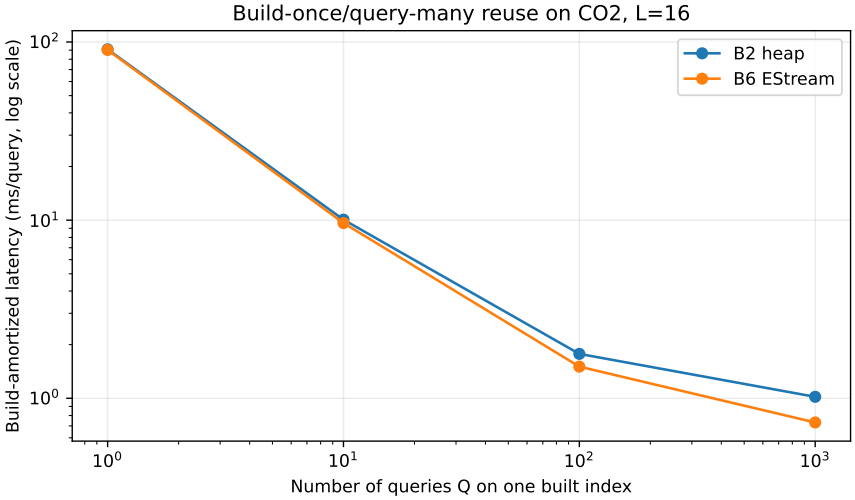


Fig. 7 Build-amortized latency under repeated queries on one CO_2 index.

one index but do *not* replay a single $(K, h, \ell_{\max})$ triple: a deterministic 1,000-query workload cycles $K \in \{10, 50, 200, 500\}$, $h \in \{n, n/2, n/4, n/16\}$, and $\ell_{\max} \in \{8, 12, 16\}$, and each reported Q is a nested prefix of that mixed stream. A one-query workload is therefore construction dominated: build-amortized latency is 90.947 ms/query for B2 and 90.254 ms/query for B6 (1.01×). At $Q = 10$ the ratio is 1.04×; at $Q = 100$ it rises to 1.18×; and at $Q = 1,000$ the costs are 1.019 and 0.732 ms/query, respectively, a 1.39× advantage. Query-only throughput for the 1,000-query mixed batch is approximately 1,076 queries/s for B2 and 1,558 queries/s for B6. The result supports the stated execution model: the structural index is useful when it is reused, while one-shot use is dominated by construction.

9 Discussion

9.1 Novelty boundary and interpretation

The paper does not claim that a capacity-K heap, QuickSelect, K-way merge, ranked-stream heads, or the Threshold Algorithm principle is itself new. Nor does it claim a new production OPST constructor. The defensible contribution is the OPF-specific transformation

$$\begin{aligned}\text{occurrence-equivalent OP loci} &\to \text{one shift-safe mass}\\ &\to \text{comparator-consistent stream}\\ &\to \text{head filtering} \to \text{reusable top-}K\text{ merge},\end{aligned}$$

together with a formal machine-arithmetic contract and a fair Java implementation. The shift-safe mass closes a concrete $0 \cdot \infty$ intermediate failure mode of the positive-exponential factorization. The baseline suite then separates that stream-level scoring benefit from generic top-K selection machinery. This narrower positioning is important: novelty lies in making endpoint forgetting structurally reusable on occurrence-equivalent OP index paths, not in renaming standard heap or merge operations.

9.2 When EStream-OPF helps

The real-data campaign rejects an unconditional speedup claim. EStream-OPF pays for stream annotation, head selection, cursor state, and a merge queue. It tends to benefit when occurrence-equivalent chains are long enough to amortize one shifted mass, K/E is small enough for aggressive stream filtering, and the same index is reused. Conversely, shallow compression or unfavorable stream/head distributions can favor B2 or B4. The 500k Household row is particularly informative because more than 99.99% of stream heads are discarded but B2 and B6 are at practical parity: pruning rate alone is not a sufficient cost predictor. B4 can also be fastest because it captures factorization savings without paying B6's head/cursor/merge overhead.

A descriptive rule fitted to this campaign, not a learned planner, is:

- use **B4** as the default factorization selector when loci/stream is in roughly $[2.5, 3.1]$ (NASDAQ, SILSO, Household 100k);
- use **B6** when loci/stream ≥ 3.1 and n is large enough that K/E is small (CO_2, S&P 500, Air Quality), expecting B4 to remain within about 6%;
- avoid B6 when loci/stream < 2.5: prefer B4 on modest E (El Niño) and B2 on a huge shallow index (Household 500k).

Nile is a cautionary tiny-n row: compression is high, but $K/E = 0.34$ and all medians sit in a 0.03–0.06 ms band, so ranking algorithms there is not a scalability result. These observations motivate a future adaptive planner that chooses B2, B4, or B6 using loci/stream, estimated $Z_{\text{stream}}/Z_{\text{locus}}$, K/E, and recent query history.

9.3 Generalization boundary of the factorization

The rightmost-start shift is exact because exponential decay is multiplicatively separable over additive ages: $e^{-k(x+y)} = e^{-kx}e^{-ky}$. The same algebra extends to discrete geometric decay $\rho^{x+y} = \rho^x\rho^y$ and, more generally, to positive decay functions with the required multiplicative semigroup property. Linear, polynomial, and logarithmic decay do not generally satisfy this identity, so they cannot reuse Eq. (8) without approximation, alternative bounds, or additional per-locus state. This boundary both clarifies why the present theorem is specific to endpoint exponential forgetting and identifies a principled route for future decay models.

9.4 Weak ties and numerical meaning

Weak-order semantics preserves equal values instead of silently dropping them. This matters for integer, quantized, and sensor series. The CO_2 preprocessing snapshot contains a substantial fraction of tied windows, and the controlled generator reaches tie rates above 80%. The canonical dense-rank identity plus deterministic predecessor–successor tie key therefore define a reproducible query universe. The binary64 contract is shift-safe and deterministic, but it is not a proof of arbitrary-precision cross-stream ordering for adversarial near ties. All reported correctness tests compare against the same declared machine order, and a separate 80-digit Decimal oracle checks controlled exact-real cases.

9.5 Threats to validity and limitations

First, the bounded reference builder enumerates weak-order windows only up to a chosen L and is designed for independent reproduction, not for the asymptotic construction guarantees of production OPST work [10, 11]. Consequently, the 500k experiment stress-tests the Java query layer under a large bounded index but does not establish multi-million-scale end-to-end indexing superiority. Second, the ten real benchmark instances cover several domains but are not independent in every row: the two Household instances are nested prefixes of one source product. We therefore report a nine-product statistical sensitivity analysis as well as the ten-instance summary. Third, Java timings can vary with JIT compilation, garbage collection, CPU scheduling, and container contention; warmups and IQR reporting reduce but do not eliminate this threat, and the present study uses one machine configuration with a fixed B1–B6 execution order. Independent timing campaigns of the same (n, L, K, h) triple must not be pooled. Fourth, the experiment runner did not retain per-repetition timing samples or JFR/RSS allocation traces, so we do not claim run-level significance or byte-level memory savings; the reported Wilcoxon analysis operates on dataset medians, and structural counters are used only to explain work reduction. Fifth, naive shifted-mass summation removes the catastrophic product failure but does not provide arbitrary-precision accumulation. Finally, performance conclusions remain conditional on K, L, forgetting horizon, compression, and data distribution.

10 Conclusion

Shift-Safe EStream-OPF formulates repeated forgetting-aware order-preserving ranking as an indexed global top-K query. The mathematical core anchors each nonempty occurrence-equivalent stream at its rightmost start $a = \max A$,

$$\mathrm{fsup}_k(A, d) = e^{-k(n-(a+d-1))} \sum_{s \in A} e^{-k(a-s)},$$

so every exponential argument is nonpositive and the shifted mass contains an exact unit term. Formal weak-order equivalence, maximal stream partition, a strict total comparator, rounded-plateau ordering, K-head sufficiency, and an induction-based merge proof form a complete correctness chain from indexed loci to the returned machine ranking. Explicit Java bounds show why the shifted mass remains finite in the artifact and why an overflowed age term becomes deterministic $-\infty$ rather than NaN.

The expanded Java 21 evidence strengthens and narrows the empirical claim. B6 has lower median query latency than the competitive capacity-K heap B2 on 8 of 10 real benchmark instances at the core setting, including B2/B6 ratios of 1.72× on weekly CO_2, 1.69× on SILSO daily sunspots, 1.64× on UCI Household 100k, and 1.43× on NASDAQ Composite. Factorization-only B4 is the fastest method on four rows and stays within 6% of B6 on the four rows where B6 wins, so sharing one stream mass is the main portable saving and K-head pruning is optional. El Niño favors B4 over both B2 and B6, and Household 500k is at practical B2/B6 parity. The independent CO_2 length sweep moves from a B2 advantage at $L = 8$ to B6 advantages at $L \geq 12$, while build amortization on a mixed 1,000-query workload raises B2/B6 from 1.01× for one query to 1.39× for 1,000 queries. Cross-instance median analysis has a geometric-mean B2/B6 ratio of 1.278; removing the nested Household 100k prefix yields 1.243. These conditional results are scientifically more informative than an unconditional speedup headline.

Future work should attach the query layer to a production OPST and evaluate multi-million-length series end to end; retain raw JMH/JFR measurements for platform-replicated latency, allocation, and memory analysis; derive an adaptive B2/B4/B6 planner; investigate compensated summation and certified interval/high-precision fallback for near-equal scores; and develop bounds for non-exponential recency models that do not possess multiplicative separability. Further extensions include dynamic index maintenance for appended observations, multivariate or multi-sensor weak-order representations, and downstream studies in anomaly screening and interactive temporal retrieval where repeated recency-aware ranking is operationally meaningful.

Supplementary information. The Java 21 reproducibility artifact, processed benchmark CSVs, tests, and dataset provenance are available at https://github.com/Porcupine1805/ESTREAM-OPF.

Acknowledgements. During manuscript preparation and revision, the authors used OpenAI ChatGPT and Codex for language editing, code and document assistance, literature organization, experiment-table consistency checking, and manuscript formatting. These systems were not treated as authoritative scientific sources. The authors

independently checked the mathematical arguments, references, source code, data preprocessing, executed experiments, and interpretations, and they take responsibility for the final content. Reported experimental numbers come from executed programs and the supplied result files rather than model-generated values. No generative-AI system is listed as an author, and no generative-AI images are used; the figures are generated from the reported experimental data.

Declarations

- **Funding.** The authors did not receive support from any organization for the submitted work.
- **Conflict of interest/Competing interests.** The authors declare that they have no competing interests.
- **Ethics approval and consent to participate.** Not applicable. This work uses public time-series datasets and does not involve human participants or animals.
- **Consent for publication.** Not applicable.
- **Data availability.** All benchmark series are public. Provenance, preprocessing, and processed CSVs are recorded in the companion repository https://github.com/Porcupine1805/ESTREAM-OPF (`DATASET_SOURCES.md`). Latency tables in this manuscript are grounded in `publication_matrix.csv` in that repository.
- **Materials availability.** Not applicable.
- **Code availability.** A Java 21 reproducibility artifact (B1–B6, tests, Decimal oracle, and benchmark runners) is available at https://github.com/Porcupine1805/ESTREAM-OPF. Upon acceptance, the artifact will also be deposited in a DOI-minting archive.
- **Author contribution.** Khac-Chien Nguyen: conceptualization, methodology, software, experiments, writing—original draft. Nhut-Minh Bui: software, validation, writing—review and editing. Cong-Phat Tran: data curation, experiments, writing—review and editing. All authors read and approved the manuscript.

References

[1] Kim, J., Eades, P., Fleischer, R., Hong, S.-H., Iliopoulos, C.S., Park, K., Puglisi, S.J., Tokuyama, T.: Order-preserving matching. Theoretical Computer Science **525**, 68–79 (2014) https://doi.org/10.1016/j.tcs.2013.10.006

[2] Crochemore, M., Iliopoulos, C.S., Kociumaka, T., Kubica, M., Langiu, A., Pissis, S.P., Radoszewski, J., Rytter, W., Walen, T.: Order-preserving incomplete suffix trees and order-preserving indexes. In: String Processing and Information Retrieval (SPIRE). Lecture Notes in Computer Science, vol. 8214, pp. 84–95. Springer, ??? (2013). https://doi.org/10.1007/978-3-319-02432-5_13

[3] Wu, Y., Hu, Q., Li, Y., Guo, L., Zhu, X., Wu, X.: Opp-miner: Order-preserving sequential pattern mining for time series. IEEE Transactions on Cybernetics **53**(5), 3288–3300 (2023) https://doi.org/10.1109/TCYB.2022.3169327

[4] Wu, Y., Zhao, X., Li, Y., Guo, L., Zhu, X., Fournier-Viger, P., Wu, X.: Opr-miner: Order-preserving rule mining for time series. IEEE Transactions on Knowledge and Data Engineering **35**(11), 11722–11735 (2023) https://doi.org/10.1109/TKDE.2022.3224963

[5] Li, Y., Liu, J., Guo, Y., Liu, J., Wu, Y.: Aop-miner: Approximate order-preserving pattern mining for time series. In: 2022 IEEE International Conference on Knowledge Graph (ICKG), pp. 149–156 (2022). https://doi.org/10.1109/ICKG55886.2022.00026

[6] Wu, Y., Meng, Y., Li, Y., Guo, L., Zhu, X., Fournier-Viger, P., Wu, X.: Copp-miner: Top-k contrast order-preserving pattern mining for time series classification. IEEE Transactions on Knowledge and Data Engineering **36**(6), 2372–2387 (2024) https://doi.org/10.1109/TKDE.2023.3321749

[7] Wu, Y., Wang, Z., Li, Y., Guo, Y., Jiang, H., Zhu, X., Wu, X.: Co-occurrence order-preserving pattern mining with keypoint alignment for time series. ACM Transactions on Management Information Systems **15**(2), 1–27 (2024) https://doi.org/10.1145/3658450

[8] Wu, Y., Lou, S., Li, Y., Guo, L., Fournier-Viger, P., Wu, X.: Outo-miner: Detecting outlying occurrences in maximal frequent order-preserving patterns in time series. Information Sciences **720**, 122497 (2025) https://doi.org/10.1016/j.ins.2025.122497

[9] Zhang, Y., Wu, Y., Li, Y.: Sopp-miner: Stable distribution order-preserving pattern mining for time series. IEEE Transactions on Knowledge and Data Engineering **38**(8), 5406–5419 (2026) https://doi.org/10.1109/TKDE.2026.3692778

[10] Li, L., Zuba, W., Loukides, G., Pissis, S.P., Matsangidou, M.: Scalable order-preserving pattern mining. In: 2024 IEEE International Conference on Data Mining (ICDM), pp. 211–220 (2024). https://doi.org/10.1109/ICDM59182.2024.00028

[11] Li, L., Zuba, W., Loukides, G., Pissis, S.P., Matsangidou, M.: Order-preserving pattern mining and comparison via indexing. IEEE Transactions on Knowledge and Data Engineering, 1–18 (2026) https://doi.org/10.1109/TKDE.2026.3710634

[12] Li, Y., Ma, C., Gao, R., Wu, Y., Li, J., Wang, W., Wu, X.: Opf-miner: Order-preserving pattern mining with forgetting mechanism for time series. IEEE Transactions on Knowledge and Data Engineering **36**(12), 8981–8995 (2024) https://doi.org/10.1109/TKDE.2024.3438274

[13] Zhou, Y., Wang, F., Tang, Y.: Order-preserving pattern matching: Review. IEEE Transactions on Knowledge and Data Engineering **38**(3), 1885–1904 (2026) https://doi.org/10.1109/TKDE.2025.3649808

[14] Wu, Y., Tian, S., Li, Y., Liu, J., Li, J., Wang, W., Wu, X.: Oip-miner: One-off incremental sequential pattern mining with forgetting factor. Science China Information Sciences **68**(10) (2025) https://doi.org/10.1007/s11432-024-4496-9

[15] Wang, J., Han, J., Lu, Y., Tzvetkov, P.: Tfp: An efficient algorithm for mining top-k frequent closed itemsets. IEEE Transactions on Knowledge and Data Engineering **17**(5), 652–663 (2005) https://doi.org/10.1109/TKDE.2005.81

[16] Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. Journal of Computer and System Sciences **66**(4), 614–656 (2003) https://doi.org/10.1016/S0022-0000(03)00026-6

[17] Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-k query processing techniques in relational database systems. ACM Computing Surveys **40**(4), 1–58 (2008) https://doi.org/10.1145/1391729.1391730

[18] Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), pp. 346–357 (2002). https://doi.org/10.1016/B978-155860869-6/50038-X

[19] Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. SIAM Journal on Computing **31**(6), 1794–1813 (2002) https://doi.org/10.1137/S0097539701398363

[20] Metwally, A., Agrawal, D., El Abbadi, A.: Efficient computation of frequent and top-k elements in data streams. In: Database Theory – ICDT 2005. Lecture Notes in Computer Science, vol. 3363, pp. 398–412 (2005). https://doi.org/10.1007/978-3-540-30570-5_27 . LNCS copyright year 2004; conference held 2005

[21] Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd edn. Society for Industrial and Applied Mathematics, ??? (2002). https://doi.org/10.1137/1.9780898718027

[22] Blanchard, P., Higham, D.J., Higham, N.J.: Accurately computing the log-sum-exp and softmax functions. IMA Journal of Numerical Analysis **41**(4), 2311–2330 (2021) https://doi.org/10.1093/imanum/draa038

[23] Seabold, S., Perktold, J.: statsmodels: Econometric and statistical modeling with python. In: Proceedings of the 9th Python in Science Conference, pp. 92–96 (2010). https://doi.org/10.25080/Majora-92bf1922-011

[24] statsmodels developers: Nile River flows at Ashwan 1871–1970. statsmodels dataset documentation. Accessed September 2026. https://www.statsmodels.org/stable/datasets/generated/nile.html

[25] statsmodels developers: Yearly sunspots data 1700–2008. statsmodels dataset documentation. Accessed September 2026. https://www.statsmodels.org/stable/

datasets/generated/sunspots.html

[26] statsmodels developers: El Nino – Sea Surface Temperatures. statsmodels dataset documentation. Accessed September 2026. https://www.statsmodels.org/stable/datasets/generated/elnino.html

[27] statsmodels developers: Mauna Loa Weekly Atmospheric CO2 Data. statsmodels dataset documentation. Accessed September 2026. https://www.statsmodels.org/stable/datasets/generated/co2.html

[28] S&P Dow Jones Indices LLC: S&P 500 [SP500]. FRED, Federal Reserve Bank of St. Louis. Accessed September 2026. https://fred.stlouisfed.org/series/SP500

[29] Vito, S.: Air Quality [Dataset]. UCI Machine Learning Repository (2008). https://doi.org/10.24432/C59K5F . https://archive.ics.uci.edu/dataset/360/air+quality

[30] Nasdaq, Inc.: NASDAQ Composite [NASDAQCOM]. FRED, Federal Reserve Bank of St. Louis. Accessed September 2026. https://fred.stlouisfed.org/series/NASDAQCOM

[31] WDC-SILSO, Royal Observatory of Belgium: International Sunspot Number Version 2.0, Daily Total Sunspot Number. Accessed September 2026. https://doi.org/10.24414/qnza-ac80 . https://www.sidc.be/SILSO/DATA/SN_d_tot_V2.0.csv

[32] Hebrail, G., Berard, A.: Individual Household Electric Power Consumption [Dataset]. UCI Machine Learning Repository (2006). https://doi.org/10.24432/C58K54 . https://archive.ics.uci.edu/dataset/235/individual+household+electric+power+consumption