Mathscapes®

Lazy evaluation for streaming statistics

Iterflow connects established online statistics with lazy JavaScript generators. The benefit is demand-driven work: a consumer can stop the entire chain. The interface concept below makes that control flow inspectable, using an exact six-value example rather than a product screenshot.

Stop after three results; leave the rest unread

Research interface concept · static trace

```
from(values).filter(x => x > 5).streamingMean().take(3).toArray()
```

SOURCE	FILTER	RUNNING MEAN	LIMIT	COLLECT
5 / 6 read	3 accepted	n = 3	3 / 3 outputs	complete
[4,7,5,9,6,8]	x > 5	μ = 22/3	take(3)	[7,8,7.3333]

SOURCE TRACE

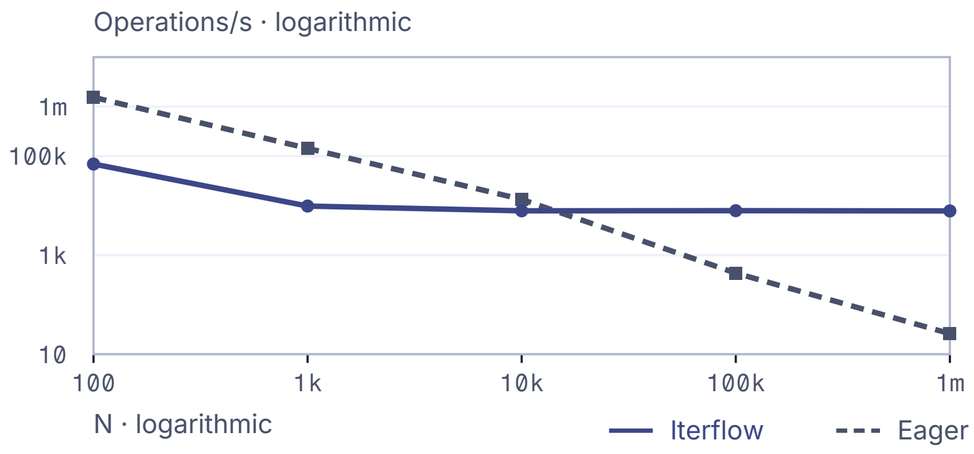
4	7	5	9	6	8
dropped	mean 7	dropped	mean 8	mean 7.3333	NOT READ

Demand stops at output 3; source value 8 remains unread.

DOI 10.5281/zenodo.18610143

Throughput with early termination

Manuscript benchmarks · not rerun



Limited consumer: take(1000)

N = 100

69,268 / 1,538,911 ops/s

All inputs needed

N = 1,000,000

7,818 / 26 ops/s

Consumer stops early

Rates compare Iterflow / eager array chain

DOI 10.5281/zenodo.18610143

01 / HOW IT WORKS

State and memory requirements

A terminal pulls through the chain; take(n) is a lazy limiting transform. Filter selectivity controls source reads, not source length alone. Mean, variance, EWMA, covariance, correlation and prior-baseline z-score retain scalar state. Windows and extrema retain O(k); collection and Quickselect median retain O(n). Median has average-case linear time. Array.shift() in the described extrema implementation can cause O(nk) work, unlike an ideal true deque.

02 / WHAT THE PAPER MEASURES

Benchmark results

Other benchmarks ask different questions. Prefix correlation takes 0.66 versus 518 ms for 10,000 paired values: streaming avoids recomputing every prefix. At 100,000 values, mean yields 8,022 versus 1,444 operations/s; variance yields 1,815 versus 691 against the selected array baselines. Conversely, generator EWMA gives 213 versus 951 operations/s for a handwritten loop, and z-score gives 175 versus 594. Efficient state does not eliminate per-element composition overhead.

03 / WHERE IT FITS

Implementation and limitations

Windowed variance uses six lines versus twelve, yielding 9,705 versus 87,870 operations/s for imperative @stdlib at 100,000 inputs. Algorithms and variance denominators differ. Native helpers offer transforms; incremental libraries offer accumulators. Reported Tinybench results use Node 22, ARM64 Linux and synthetic data, not rerun here. Benchmark metadata are incomplete. Types do not validate runtime values; z-score starts with two NaNs. Async streams, quantiles and fusion remain proposed.