Author
Sem Lelie

Supervisors
Ricardo Marroquim
Responsible Professor
Arthur Avramiea
Daily Supervisor

Affiliation
EEMCS, TU Delft
VU Amsterdam

Minimising Data-Layout & Copy Overhead in Python EEG Analysis Pipelines

EEG biomarker pipelines look compute-bound, but much of their cost is **moving arrays**, not computing statistics.
Can we remove that overhead without changing a single scientific output?

Course
CSE3000 Research Project

Contact
s.lelie@student.tudelft.nl

Code & data
reproducible benchmark harness, fixed seeds

Date
June 2026

01 Why it is costly: a deep copy that doubles memory

Between every stage, the **NBT** EEG toolbox defensively **deep-copies** its whole data container. That copy, not the science, is the hidden cost.

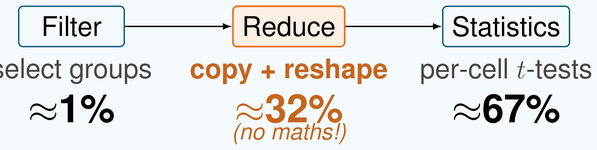
A deep copy is **expensive in two ways**: it is slow ($\mathcal{O}(N\,C\,F)$), *and* it allocates a full duplicate, so it **doubles the stage's peak memory**.

On memory-bound hardware the memory is what bites:
- Modern pipelines are bound by **memory bandwidth**, not arithmetic.
- A **cohort sweep** runs many pipelines at once; the doubling tips them into **swap**, stalling a fast run by $\sim 10\times$ (its tail latency).
- Yet this layout/copy cost is rarely measured in domain-science toolchains.

02 Where it hides: a reduce stage that does no maths

NBT runs three stages. Profiling shows the middle *reduce* stage performs **no arithmetic at all**: its entire cost is a deep copy plus a reshape, pure data movement. That is the stage we target.

we target this stage



Share of baseline end-to-end runtime ($N = 10.000$, two sessions/subject).

03 The fix: share the buffer, do not copy it

Once a stage has written its arrays they never change, so the defensive deep copy is wasted work: each stage can simply *point at* the shared buffer instead of cloning it. We test three independent strategies (the full 2^3 factorial):

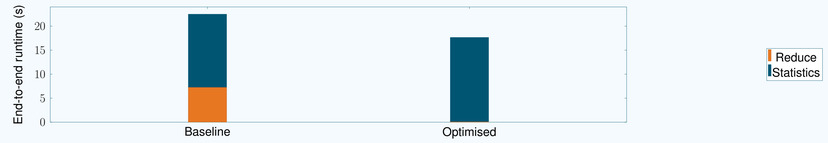
A Zero-copy views – share the buffer, no duplicate. *(the workhorse)*
B Layout pinning – build the repeat-session buffer contiguous for the kernels.
C Lazy materialisation – defer that buffer until something reads it.
B and C act only when subjects have ≥ 2 recording sessions.

Identical science: **126 / 126 full-pipeline equivalence checks pass.**

Bench: Intel i7-13700H, 16 GB; synthetic container $N \times 64 \times 50$; 30 paired runs; Wilcoxon signed-rank.

04 Result: removing the copy empties the stage

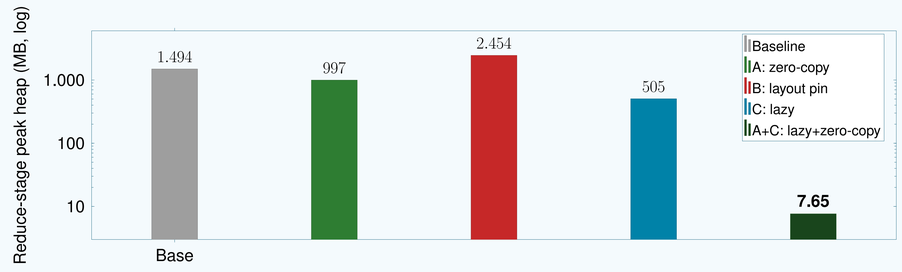
The reduce stage was almost all copy, so zero-copy nearly erases it.



$N = 10.000$, $M = 2$ (filter step $<1\%$, omitted). Reduce: $7.24 \rightarrow 0.075$ s ($97\times$). The statistics step is pure arithmetic and out of scope, so it caps the end-to-end *gain at $1.3\times$ (Amdahl) – the limit is that stage, not the optimisation.*

05 The big win is memory: lazy collapses it, pinning backfires

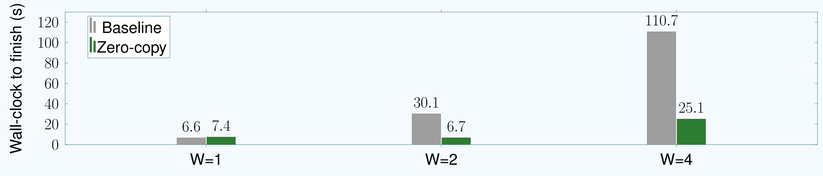
With ≥ 2 sessions a large expansion buffer dominates; deferring it (lazy) collapses peak memory.



*$N = 10.000$, $M = 2$ (A=zero-copy, B=layout, C=lazy). Lazy needs zero-copy: **A+C** cuts peak heap to 7.65 MB ($195\times$); at a single session zero-copy alone already gives $306\times$. **Layout (B) backfires** – two buffers push it* above *the baseline.*

06 No memory doubling means no swap under load

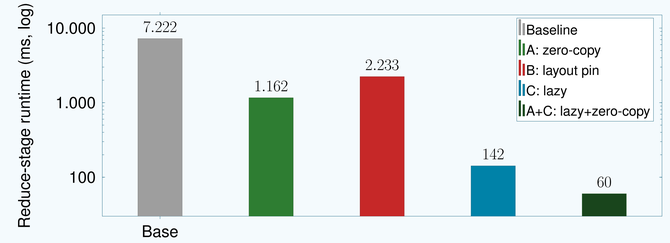
A cohort sweep runs many pipelines at once; the baseline's duplicates exhaust RAM and force swapping.



4 workers, $N = 100.000$: the baseline swaps (110.7 s); zero-copy never duplicates and finishes in 25.1 s ($4.4\times$).

07 Runtime too: every strategy beats the baseline

At ≥ 2 sessions every candidate cuts reduce-stage time below the baseline, all highly significant; lazy+zero-copy nearly eliminates the stage.



$N = 10.000$, $M = 2$. Reduce-stage runtime per strategy; lazy+zero-copy (A+C) brings it to 60 ms. ***Every candidate beats the baseline on all 30 paired runs*** *($p \approx 1.9 \times 10^{-9}$, Wilcoxon signed-rank). Layout pinning (B) is faster on time but costs memory (block 05).*

08 The payoff at a glance

Same science, a fraction of the memory and time.

$306\times$ smaller peak memory (single session)	$195\times$ smaller peak heap (repeat sessions)
$97\times$ faster reduce stage (the copy is gone)	$4.4\times$ throughput under parallel load (no swap)

The reduce stage is, in effect, eliminated.

09 What to take away

1. **Remove eager deep copies first.** A deep copy's worst cost is its *memory*, not its time – that is what separates fitting in RAM from swapping under load.
2. **Judge the win by the right metric:** memory, tail latency, and parallel throughput ($4.4\times$, no swap), not mean single-run time.
3. **Apply each trick where it acts.** Lazy materialisation is decisive once sessions repeat; layout pinning backfires – neither helps a single session.

Next: the statistics step is now the bottleneck. **Vectorising** its per-cell t-test loop is the route to turning these stage-level gains into end-to-end ones.

TU Delft