# unclone: Performance and Numerical Agreement with PyClone-VI

kojix2

2026-08-12

## Abstract

We evaluated unclone (kojix2 2025a, 2025b), a reimplementation of PyClone-VI (Gillis and Roth 2020) with a Rust kernel, on a workstation with an 8-core/16-thread AMD Ryzen 7 5700X. When both implementations started from the same NumPy-generated initial values, all 18 dataset–seed combinations produced identical mutation partitions (adjusted Rand index 1.0). The largest absolute difference in cellular prevalence (CCF) was $5.5 \times 10^{-13}$. At one thread, unclone was 2.8–5.9 times faster than PyClone-VI on inputs of 100–2,440 mutations. The unclone time includes CLI startup, whereas PyClone-VI was measured in a warmed Python process. On the full TRACERx input, the unclone kernel ran 4.80 times faster on eight physical cores and 5.10 times faster on 16 hardware threads than on one thread. PyClone-VI gained at most 1.07 times. At eight threads, the full-input latency was 0.084 s for the unclone CLI and 1.723 s for warmed in-process PyClone-VI. The median peak resident memory was 33.4 MB and 280.1 MB, respectively.

## Methods

### Implementations and data

unclone was built with `make build release=1 cpu=native`. The base commit and the SHA-256 hash of the measured Rust source diff are recorded in `data/environment.json`. PyClone-VI 0.2.0 was installed from upstream commit `07306831a9a4`. The official synthetic and TRACERx example files were used as input data. PyClone-VI discards mutations that are not observed in every sample. The TRACERx input was filtered once to the 2,440 complete mutations, and this filtered input was used for both implementations. For the scaling measurements, smaller inputs were created containing the first 100, 250, 500, or 1,000 of these mutations. The SHA-256 hashes of all generated inputs are also recorded in `data/environment.json`.

The unclone kernel uses the same Rayon execution path at every thread count. Variational contractions and the initial expected log likelihood share the same matrix-multiplication kernels. Beta-binomial terms that are constant across the CCF grid are evaluated once per observation.

Both implementations used the same settings: beta-binomial density, 40 clusters, 100 CCF grid points, precision 200, convergence tolerance $10^{-6}$, at most 10,000 iterations, one restart, and seed 7. For the numerical comparison, seeds 42 and 123 were also used with unclone's `--python-compatible` initialization.

### Timing protocol

Before PyClone-VI was timed, an unrecorded warm-up was run to compile the Numba functions. PyClone-VI was then measured in a single persistent process. Its timer covers input loading, likelihood construction, initialization, and inference, but excludes result serialization. Both the BLAS and Numba thread counts

were set to the requested value. unclone was launched as a fresh CLI process for every repetition. For the cross-tool comparison, the complete CLI wall time was used, including input and output. For parallel scaling, the kernel-only time reported at the Rust profiling boundary was used, so that fixed CLI overhead does not affect the scaling measurement.

Each timing condition was repeated seven times in a deterministic randomized order. Medians and interquartile ranges (IQR) are reported. Runs with one to eight threads were pinned to distinct physical cores 0 through $n-1$. The 16-thread run used both hardware threads of all eight cores. The peak resident set size (RSS) was measured in three fresh processes with GNU `time`. All raw observations, summaries, and the benchmark driver are available in `paper/data` and `paper/scripts/benchmark.py`.

### Computing environment

Table 1: Measurement environment. The host had no hypervisor and was otherwise idle; the CPU frequency was not fixed.

| Item | Value |
|---|---|
| CPU | AMD Ryzen 7 5700X |
| Topology | 8 cores, 16 hardware threads, one socket |
| Memory | 60 GiB |
| OS | Linux 7.0.0-29-generic, x86-64 |
| CPU governor | `powersave` (`amd-pstate-epp`), boost enabled |
| unclone | 0.0.4, native CPU build |
| PyClone-VI | 0.2.0 |
| Rust / Crystal | 1.97.1 / 1.21.0 |
| Python | 3.14.4 |
| NumPy / Numba / SciPy | 2.5.2 / 0.67.0 / 1.18.0 |

## Numerical agreement

Six input sizes were tested with three seeds each. With matched initialization, every run gave an ARI of 1.0. The CCF correlation rounded to 1.0 in all runs, and the maximum absolute CCF difference was between $1.9 \times 10^{-14}$ and $5.5 \times 10^{-13}$ (Table 2). unclone is numerically concordant with PyClone-VI at near-floating-point precision. This result does not imply bitwise identity on every platform or input.

Table 2: Agreement with shared NumPy initialization (`data/quality.csv`).

| Dataset | Mutations | Seeds tested | Minimum ARI | Maximum $\lvert\Delta\text{CCF}\rvert$ |
|---|---|---|---|---|
| synthetic | 100 | 7, 42, 123 | 1.0 | $1.3 \times 10^{-14}$ |
| TRACERx | 100 | 7, 42, 123 | 1.0 | $1.3 \times 10^{-14}$ |
| TRACERx | 250 | 7, 42, 123 | 1.0 | $1.2 \times 10^{-13}$ |
| TRACERx | 500 | 7, 42, 123 | 1.0 | $1.5 \times 10^{-13}$ |
| TRACERx | 1,000 | 7, 42, 123 | 1.0 | $3.8 \times 10^{-13}$ |
| TRACERx | 2,440 | 7, 42, 123 | 1.0 | $5.5 \times 10^{-13}$ |

## Performance

### Single-thread comparison

At one thread, unclone was faster than PyClone-VI on every input. The unclone time includes process startup and serialization, whereas the PyClone-VI time excludes process startup, JIT compilation, and serialization. The median speedup ranged from 2.81 to 5.88 times (Figure 1 and Table 3). On the full TRACERx input, the median latency was 0.313 s for unclone and 1.838 s for PyClone-VI, with IQRs of 0.024 s and 0.012 s, respectively.

Table 3: One-thread median latency over seven runs (`data/timing_raw.csv`). U = unclone; PCV = PyClone-VI; IP = warmed in-process measurement.

| Dataset | Mutations | U CLI (s) | PCV (s) | PCV/U |
|---|---|---|---|---|
| synthetic | 100 | 0.023 | 0.092 | 3.99 |
| TRACERx | 100 | 0.027 | 0.076 | 2.81 |
| TRACERx | 250 | 0.038 | 0.177 | 4.60 |
| TRACERx | 500 | 0.078 | 0.346 | 4.43 |
| TRACERx | 1,000 | 0.170 | 0.719 | 4.24 |
| TRACERx | 2,440 | 0.313 | 1.838 | 5.88 |

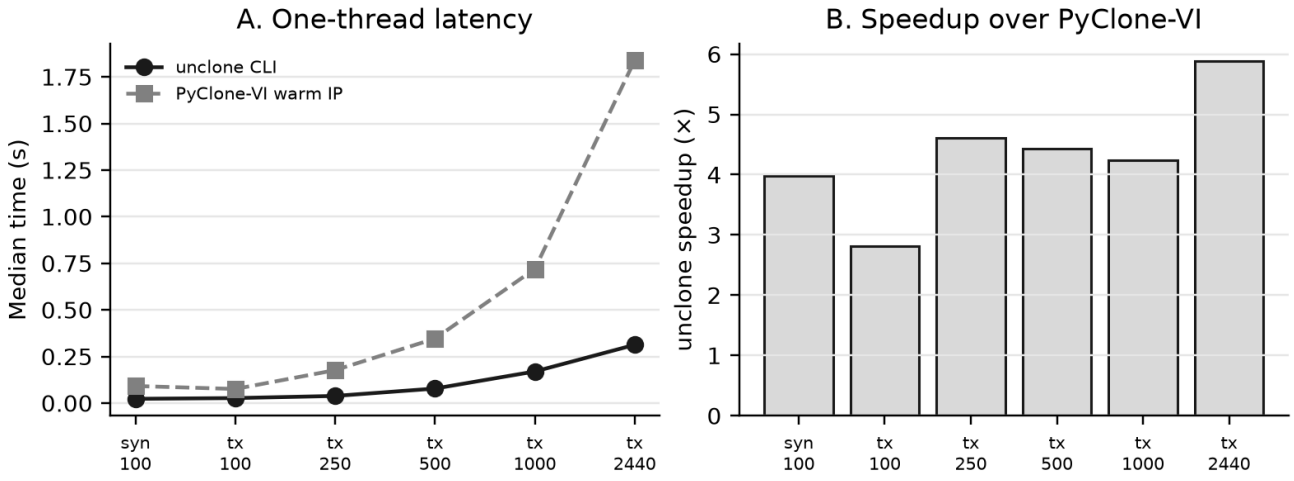


Figure 1: One-thread latency and speedup.

### Multithread scaling

On the full TRACERx input, the unclone kernel scaled with the number of threads (Figure 2). Relative to one thread, the kernel achieved speedups of 1.52, 3.16, 4.80, and 5.10 times at 2, 4, 8, and 16 threads, respectively. The parallel efficiency at eight cores was 60%. Simultaneous multithreading reduced the kernel time from 0.061 s at eight threads to 0.057 s at 16 threads, an improvement of about 6%. Performance saturates near the number of physical cores. The relative speedup is lower than in the previous implementation because the one-thread kernel time has been reduced by 56%; the absolute multithread latency also improved.

PyClone-VI showed little thread scaling. Its median latency stayed between 1.71 and 1.84 s at all thread counts. The best median, at four threads, was 1.07 times faster than at one thread. The 16-thread run

was slower at 1.81 s. This result applies to this workload and to the linked OpenBLAS and Numba versions; it may not generalize to other PyClone-VI inputs.

Table 4: Thread scaling on 2,440 TRACERx mutations (`data/thread_summary.csv`). U = unclone kernel; PCV = warmed in-process PyClone-VI.

| Threads | U kernel (s) | U speedup | PCV (s) | PCV speedup |
|---|---|---|---|---|
| 1 | 0.293 | 1.00 | 1.838 | 1.00 |
| 2 | 0.193 | 1.52 | 1.753 | 1.05 |
| 4 | 0.093 | 3.16 | 1.714 | 1.07 |
| 8 | 0.061 | 4.80 | 1.723 | 1.07 |
| 16 | 0.057 | 5.10 | 1.808 | 1.02 |

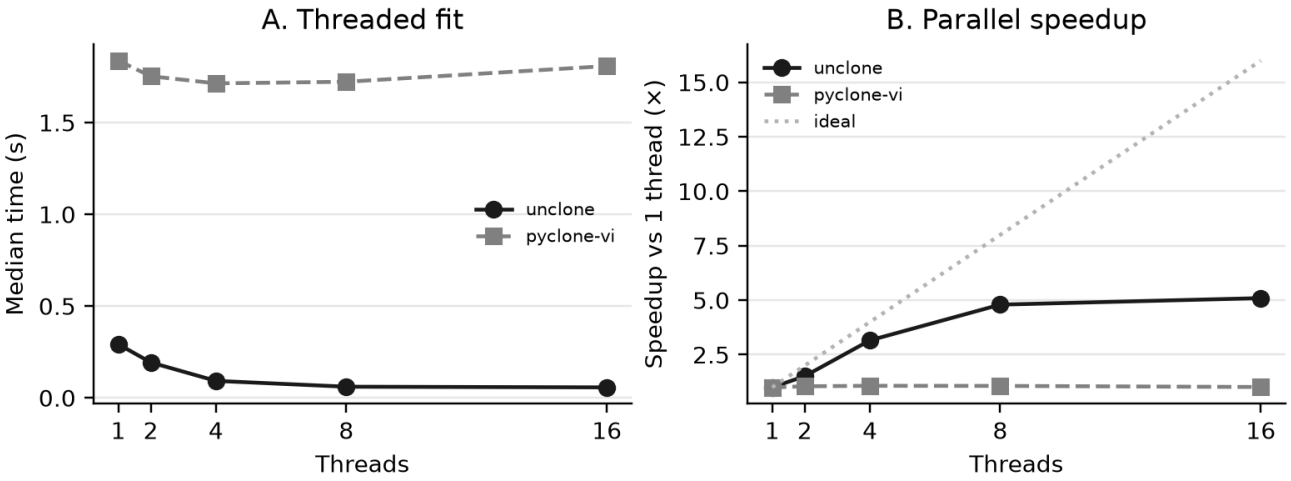


Figure 2: Fit time and parallel speedup on full TRACERx.

### Memory

At one thread, the median peak RSS was 32.7 MB for unclone and 277.0 MB for PyClone-VI, an 8.5-fold difference. At eight threads, the values were 33.4 MB and 280.1 MB, an 8.4-fold difference. Additional worker threads added little memory in either implementation on this input.

## Limitations

This study has several limitations. A single machine, a single compiler build, one upstream example family, and short-running workloads were used. Dynamic frequency scaling remained enabled during the measurements. Randomized ordering, CPU affinity, and repeated medians reduce frequency and thermal effects, but do not eliminate them. Small-input measurements have larger relative variation because their runtimes approach the process-startup duration. The timing boundaries differ between the two comparisons. The cross-tool comparison favors PyClone-VI, because it warms the JIT and excludes process startup. The parallel-scaling comparison uses narrower, implementation-specific boundaries. These results are reproducible measurements of this specific configuration, not universal, hardware-independent constants.

## Conclusions

On a local 8-core workstation, unclone reproduced the partitions and CCF estimates of PyClone-VI to near machine precision. It was 2.8–5.9 times faster at one thread and used about one eighth of the process memory. Its Rust kernel achieved a 4.80-fold speedup on eight physical cores, whereas PyClone-VI showed little thread scaling on the same workload. Hyperthreading provided a small additional gain. These measurements replace the earlier two-vCPU cloud results and characterize the multithread performance of both implementations.

## References

Gillis, Simone, and Andrew Roth. 2020. "PyClone-VI: Scalable Inference of Clonal Population Structures Using Whole Genome Data." *BMC Bioinformatics* 21 (1): 571. https://doi.org/10.1186/s12859-020-03919-2.

kojix2. 2025a. "Unclone." https://github.com/kojix2/unclone. https://github.com/kojix2/unclone.

———. 2025b. "Unclone." https://doi.org/10.5281/zenodo.20711091.