PREPRINT · NOT PEER REVIEWED

SwiftLD: an agent-built, byte-identical vcftools LD that is 826-fold faster than vcftools and 7.6-fold faster than PLINK 2

Joe Christensen (ORCID 0009-0007-0908-1299)
Translational Neuropsychiatry Unit (TNU), Department of Clinical Medicine, Aarhus University, Denmark
Correspondence: joechristensen@clin.au.dk

Abstract

SwiftLD is a drop-in replacement for the pairwise linkage disequilibrium (LD) modes of vcftools 0.1.17, `--hap-r2` and `--geno-r2`, which population-genetic pipelines use to compute r^2, D and D′ between nearby variants. On three regions of 1000 Genomes chromosome 22 with 503 to 2,504 samples it wrote the same output as vcftools, byte for byte, and was 826-fold faster (geometric mean at two threads on an AVX-512 CPU). It was also 7.6-fold faster than PLINK 2, the fastest alternative, from VCF input to LD output on all five regions tested; PLINK 2 writes a different file, although its r^2 values agree with vcftools' within 10^{-6}. One megabase with 503 samples took vcftools 29 s and SwiftLD 0.049 s, and on an ordinary AVX2 laptop SwiftLD was 102-fold faster than vcftools at 12 threads. vcftools was slow because it writes every kept site to a temporary file and, for every pair, reads the second site back one genotype byte per call, tallies the pair one haplotype at a time and flushes the output after each row. I built SwiftLD with correctness-gated coding agents (Claude Opus 5.5): every change had to keep the output byte-identical to vcftools. No output byte differed in 167,749 differential tests on three machines. Matching vcftools exposed seven upstream defects, five of which a short patch fixes.

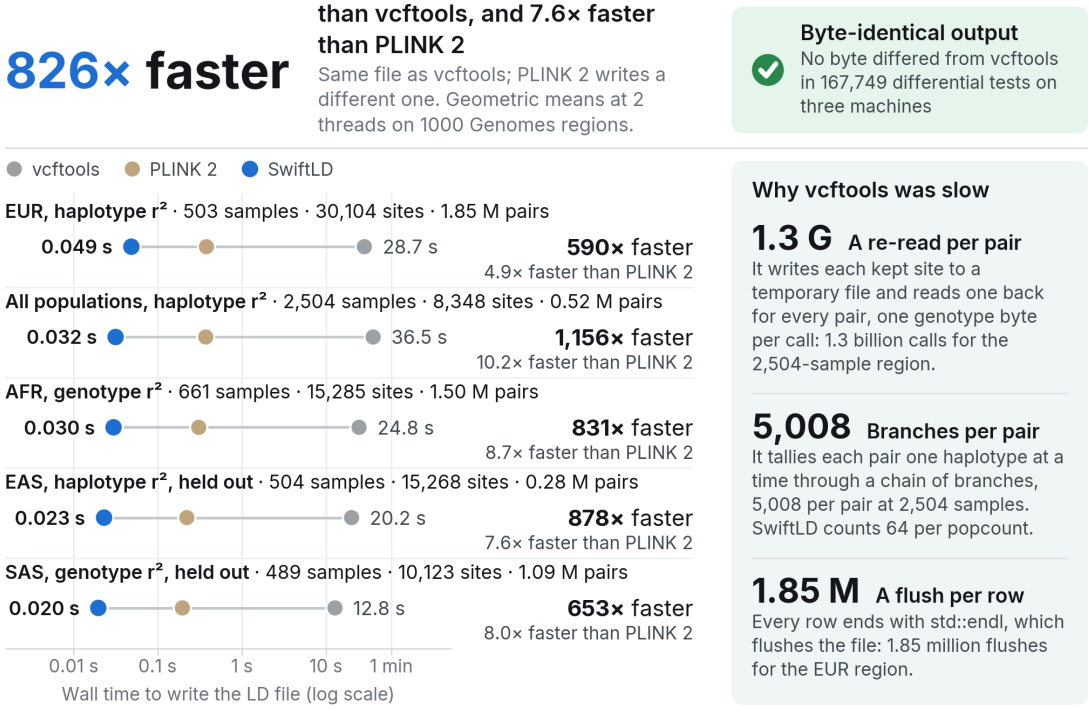


Figure 1. SwiftLD writes the same file as vcftools in a few hundredths of a second where vcftools takes 13 to 36 s, and is 4.9- to 10.2-fold faster than PLINK 2, which writes a different file. Wall time of vcftools 0.1.17 (grey; one thread), PLINK 2 (sand; import plus LD) and SwiftLD (blue), the last two at two threads, on five regions of 1000 Genomes chromosome 22, log scale; the bottom two regions were held out from development. AVX-512 cloud VM with 2 vCPUs. PLINK 2 ratios use SwiftLD times from the same benchmarking script (Section 4.5). Right: the three habits that made vcftools slow (Section 2.4).

1 Introduction

Linkage disequilibrium (LD), the non-random association of alleles at different loci, enters many population-genetic analyses: pruning markers before principal component and admixture analyses, estimating recombination and effective population size from the decay of LD with distance, and fine-mapping association signals (Slatkin 2008). vcftools (Danecek et al. 2011), cited more than 16,000 times, is a standard way to compute it. `--hap-r2` reports r^2, D and D′ from phased haplotypes (Lewontin 1964; Hill and Robertson 1968), `--geno-r2` the squared correlation of genotype dosages, and both read a VCF file directly and apply vcftools' site and individual filters. They are slow: one megabase of 1000 Genomes data with 503 samples and a 200-kb window takes 29 s. PLINK 1.9 and 2.0 (Chang et al. 2015) and scikit-allel (Miles et al. 2024) are faster, but they need an import step, define filters differently and print other rows, so switching to them changes downstream numbers.

SwiftLD writes the same output file as vcftools, byte for byte. On three 1000 Genomes regions it was 826-fold faster than vcftools at two threads, and on five it was 7.6-fold faster than PLINK 2 (Fig. 1), so a pipeline can switch without re-validating a single number. The waste in vcftools is simple. It keeps every site in a temporary file and, for every pair, reads the second site back one byte at a time and counts it one haplotype at a time. Decoding each site once into bit-planes and counting pairs with popcount gave the first 77.5-fold; making input and output as cheap as the counting gave the rest. The work also exposed seven defects in vcftools, among them a buffer overflow that aborts every LD run on hardened builds.

2 Results

2.1 SwiftLD is 826-fold faster than vcftools on 1000 Genomes data

On the three 1000 Genomes development regions SwiftLD was 826-fold faster than vcftools at two threads and 402-fold faster at one (geometric means; Fig. 1, Table 1). The largest factor came on the region with all 2,504 samples, where vcftools took 36 s and SwiftLD 0.032 s (1,156-fold), because vcftools' cost per pair grows with the number of samples (Section 2.4). On the two held-out regions, which were never used during development, SwiftLD was 878- and 653-fold faster. Across all eleven benchmark sets, which add six synthetic sets with missing and unphased calls, extra FORMAT fields, several chromosomes and 1,500 samples, it was 163- to 1,309-fold faster; the slowest is a set where 5% missing calls send every pair through the general counting path.

Table 1. SwiftLD wrote byte-identical output on every 1000 Genomes region and was 590- to 1,156-fold faster than vcftools at two threads. AVX-512 cloud VM with 2 vCPUs; vcftools 0.1.17 best of three runs; SwiftLD at two threads, median of three evaluation runs (held-out regions, one run), each the best of up to three; at one thread, one run. Mode: haplotype r^2 (`--hap-r2`) or genotype r^2 (`--geno-r2`); minor allele frequency thresholds and the full options are in Supplementary Table S1. The geometric means are over the three development regions; at two threads it is the median of the three runs' geometric means (the geometric mean of the per-region medians is 827).

Region	Samples	Mode, window	Pairs written	vcftools (s)	SwiftLD 1 thread (s)	Speed-up	SwiftLD 2 threads (s)	Speed-up
EUR	503	hap, 200 kb	1,853,366	28.67	0.0981	292	0.0486	590
All populations	2,504	hap, all pairs	515,620	36.49	0.0652	560	0.0316	1,156
AFR	661	geno, 1,000 sites	1,499,500	24.79	0.0624	397	0.0298	831
EAS, held out	504	hap, 400 kb, $r^2 \geq 0.1$	276,177	20.20			0.0230	878
SAS, held out	489	geno, 300 kb	1,090,026	12.83			0.0196	653
Geometric mean						**402**		**826**

The speed-up holds on other machines (Fig. 2). On a second cloud VM of the same type, with vcftools re-timed there, SwiftLD was 541-fold faster on the development regions at two threads; on that VM vcftools ran 11–17% faster and SwiftLD 8–51% slower than on the first. On a laptop without AVX-512 (AMD Ryzen 5 5600H, 12 threads), where the build selected the portable code paths, SwiftLD was 102-fold faster on the development regions and 89-fold faster on the held-out regions.

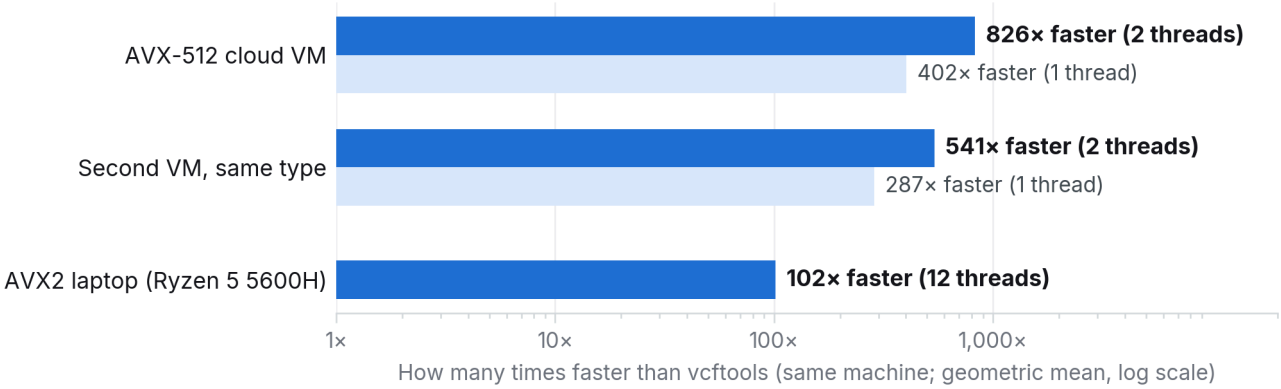


Figure 2. SwiftLD is faster than vcftools on every machine tested. Geometric mean over the three 1000 Genomes development regions of how many times faster SwiftLD was than vcftools 0.1.17 on the same machine: the AVX-512 VM of Table 1, a second VM of the same type with vcftools re-timed there, and an AVX2 laptop (AMD Ryzen 5 5600H) on which the build selected the portable code paths. Dark bars, all threads; light bars, one thread (not measured on the laptop); log scale. Every output was byte-identical to vcftools'.

2.2 The output is identical, byte for byte

No output byte differed between SwiftLD and vcftools in 167,749 differential tests (Table 2). Every code change had to pass a check that ran 300 fresh random VCF files through both programs, with random options and thread counts, and compared the output on six benchmark sets with checksums recorded for vcftools; 77 such checks add up to 23,100 random tests. Five held-out sets passed every time they were checked. On the final code, edge-case tests, among them 34- to 63-MB gzip inputs and builds without AVX-512, and separate random runs added 142,549 cases, and the full check passed on a second VM and on a laptop with a different CPU vendor. Byte identity covers row order, six-significant-digit formatting and the sign of NaN, which vcftools prints as `-nan`.

Table 2. No output byte differed in any comparison. Each test compared the complete output of SwiftLD and vcftools 0.1.17 on the same input, options and thread count; benchmark outputs were compared by sha256.

Test	Runs	Result
Random VCF files against the live vcftools during development	23,100	all identical
Held-out sets, never used during development (5 sets × 6 checks)	30	all identical
Edge-case tests on the final code (large gzip inputs, builds without AVX-512)	22,549	all identical
Separate random VCF files on the final code	120,000	all identical
Numbers formatted by SwiftLD against glibc `snprintf` with `%.6g`, vcftools' format	83,882,953	all identical
Fresh copy of the code: five 1000 Genomes outputs at 1, 2 and 3 threads	15	all identical
Full check on a second VM of the same type: random VCF files	1,500	all identical
Full check on an AVX2 laptop (AMD Zen 3): random VCF files	600	all identical

2.3 SwiftLD is 7.6-fold faster than PLINK 2, which writes a different file

SwiftLD was the fastest tool on every 1000 Genomes region (Figs. 1 and 3). From VCF input to LD output it was 4.9- to 10.2-fold faster than PLINK 2 (7.6-fold on average), 5.4- to 10.9-fold faster than PLINK 1.9 and 103- to 252-fold faster than scikit-allel. Most of PLINK 2's time goes to importing the VCF, 42–70% of its total; on data PLINK 2 had already imported, SwiftLD was still 2.2- to 4.7-fold faster than PLINK 2's LD step alone.

None of the three writes vcftools' file. PLINK 2 agreed with vcftools on every shared r^2 within 10^{-6} and scikit-allel on at least 99.96% of them, with the largest difference 4.8×10^{-6}. PLINK 1.9's `--r2` agreed on the genotype regions, but on phased data it uses an unphased composite estimator and differed by up to 0.79. All three print their own rows and columns, define allele frequency and missingness on their own imported data and do not print D′ as vcftools does, so a pipeline that switches to them has to re-validate its results (Supplementary Section S2).

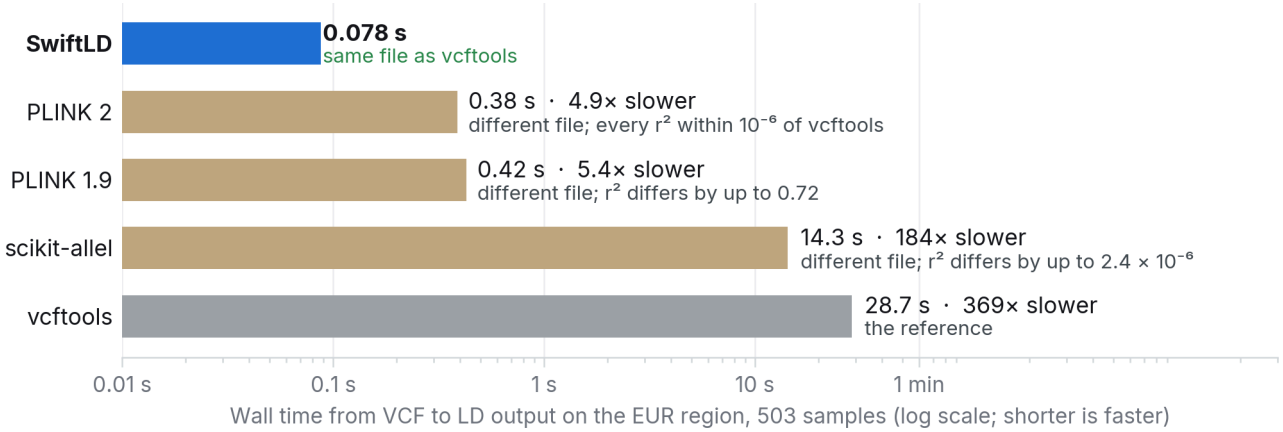


Figure 3. On the EUR region every other tool is slower than SwiftLD, and none of them writes vcftools' file. Wall time from VCF input to LD output on 1000 Genomes EUR (503 samples, 1 Mb, 1.85 million pairs), log scale; PLINK times include importing the VCF and scikit-allel's include Python start-up. PLINK at two threads on the AVX-512 VM; one script timed every tool, SwiftLD included, which is why SwiftLD's time here is 0.078 s rather than the 0.049 s of Table 1. Agreement is with vcftools' r^2 on the pairs both tools reported. Supplementary Figure S1 shows all five regions; versions are in Section 4.5.

2.4 vcftools re-reads and re-decodes a site for every pair

Three habits of vcftools account for most of its run time (Fig. 4).

It re-reads a site for every pair. vcftools writes every kept site to a temporary file, one byte per individual, and for each pair reads the second site back and decodes it into a vector of allele pairs, with one `ifstream::read` call per byte. On the 2,504-sample region that is 2,504 calls for each of 515,620 pairs, 1.3 billion in all.

It counts one haplotype at a time. It then walks the two sites haplotype by haplotype (individual by individual for `--geno-r2`) through a chain of branches and double-precision accumulators: 5,008 steps per pair at 2,504 samples.

It flushes after every row. Each output row ends with `std::endl`, which flushes the stream, 1.85 million times for the 1.85 million rows of the EUR region.

The first two make the cost of a pair proportional to the number of samples. vcftools' run time divided by the pairs it wrote was 12 to 17 µs on the regions with 489 to 661 samples and 71 µs on the region with 2,504, or 24 to 31 ns per sample and pair (Fig. 4a).

SwiftLD decodes each site once into bit-planes over the haplotypes, or over the individuals for `--geno-r2`, so the counts of a pair are popcounts of ANDed 64-bit words. It then evaluates vcftools' double-precision expressions in the same order from the exact integer counts, compiled without fused multiply-add, so every value agrees to the last bit. The first version did this and wrote rows in chunks from worker threads, strictly in order; it was 77.5-fold faster than vcftools on the six development sets (Fig. 4b). Input and output then dominated, and the next 8.0-fold came from validating and decoding regular genotype lines 64 bytes at a time with AVX-512, decompressing gzip with libdeflate (Biggers 2026) on a background thread that the parser follows, an AVX-512 formatter for vcftools' `%.6g` numbers that sends any value near a rounding tie to an exact path, and recycling the

memory pages the parser had passed. Because rows are written in order, the output does not depend on the thread count.

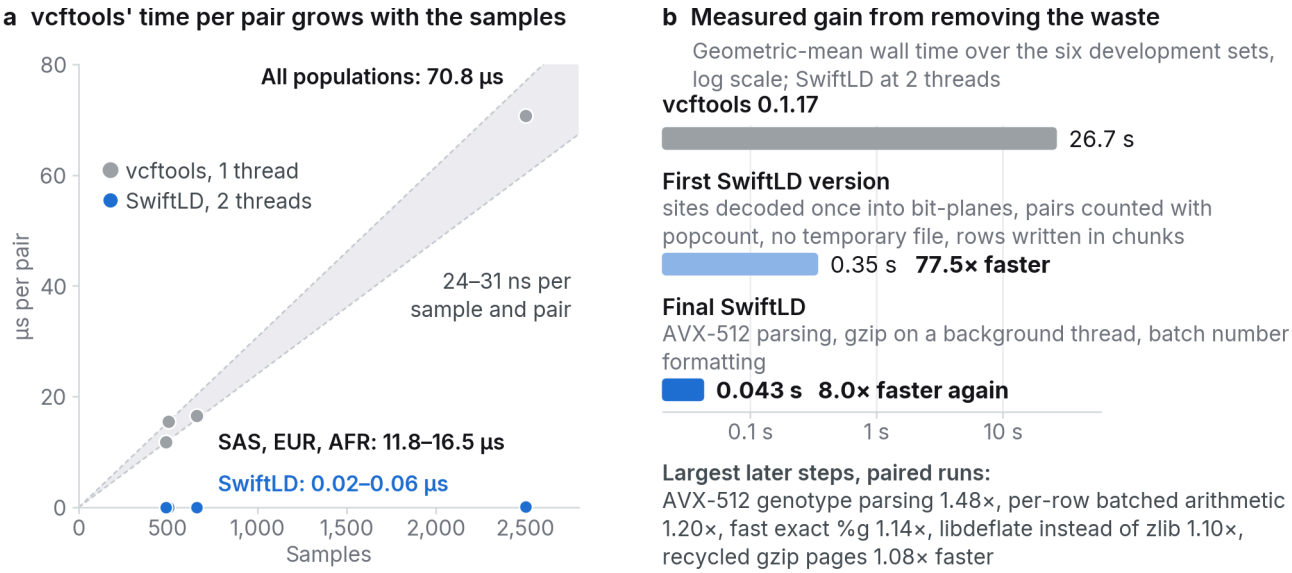


Figure 4. vcftools' cost per pair grows with the number of samples, and removing that waste gave the first 77.5-fold. (a) Wall time of vcftools 0.1.17 divided by the pairs it wrote, against the number of samples, on four 1000 Genomes regions (grey), and the same for SwiftLD at two threads (blue); the shaded wedge spans 24 to 31 ns per sample and pair. The EAS region is left out because `--min-r2 0.1` hides the pairs vcftools computed but did not print. (b) Geometric-mean wall time over the six development sets of vcftools (best of three runs), the first SwiftLD version (one run) and the final SwiftLD (median of three runs), with the largest later gains, each measured in paired runs against the version before it.

2.5 Matching vcftools exposed seven upstream defects

Fifteen functions of vcftools 0.1.17 copy a temporary-file name into a buffer one byte too short, so every `--hap-r2` and `--geno-r2` run aborts under `_FORTIFY_SOURCE`, the Ubuntu default; the reference in this work carried that fix. A zlib data error after the header, such as a wrong CRC, makes the gzip read loop run forever. "Unknown option" and three other usage errors exit with status 0. `--hap-r2` checks phase before applying `--keep` and `--remove`, so an unphased call of an excluded individual aborts the run. Sites with no called alleles pass `--maf`, contrary to the manual. A record with fewer columns than the header takes the missing samples' genotypes from the previous record, and an allele index beyond the ALT list is read as missing by the LD code but counted as called by `--freq`. The first five have patches of 3 to 15 lines, and vcftools with all five applied still writes the reference output of the SAS region byte for byte. SwiftLD reproduces every defect that shapes the output, so the two tools agree until vcftools is fixed, and it exits with an error on corrupt gzip input instead of looping. Supplementary Section S3 gives minimal reproductions and the patches.

3 Discussion

SwiftLD's margin over vcftools comes from removing work the answer does not need, not from approximation. vcftools' arithmetic is deterministic and its inputs are small integers, so its own expressions, fed exact counts, reproduce every bit. The margin over PLINK 2 is smaller, 4.9- to 10.2-fold, because PLINK 2 already counts pairs with bit operations; most of its remaining time goes to importing the VCF.

Byte identity is a stronger promise than agreement on a test set, and it is the promise that lets a user switch without re-validation. Testing every change against vcftools made it checkable while the code changed. It forced SwiftLD to reproduce vcftools' filter order, its window scan, the sign of its NaN and the defects that shape its output, and it forced an exact fallback behind every fast path. The

protection held on data the tests had never seen: on held-out regions, on a second VM and on a laptop with a different CPU vendor and no AVX-512. The same approach carries over to any tool whose output can be compared byte for byte.

Limitations. The headline was 826-fold on the VM used for development, where single runs varied by up to 20%, and 541-fold on a second VM of the same type. It uses two threads, all CPUs of the VM, against single-threaded vcftools, and scaling beyond two threads was not measured. Timed inputs were single-member gzip regions of 0.25 to 1 Mb; whole chromosomes and BGZF input, which SwiftLD reads through a slower non-streaming path, were not timed. Identity with vcftools rests on differential tests over the 18 supported options, not on proof, and it includes vcftools' defects.

A SwiftLD output file is a vcftools output file, so every pruning list, decay curve and fine-mapping input built from it stays as it was. The 29-second vcftools run on one megabase of EUR haplotypes takes SwiftLD a twentieth of a second, and on an ordinary laptop without AVX-512 SwiftLD was 71- to 117-fold faster than vcftools on each of the five regions.

4 Methods

4.1 Target

I took the pairwise LD modes of vcftools 0.1.17 (commit 1c53c3c) as the target. SwiftLD accepts 18 vcftools options in any combination and order (`--vcf`, `--gzvcf`, `--out`, `--stdout`; `--ld-window`, `--ld-window-min`, `--ld-window-bp`, `--ld-window-bp-min`, `--min-r2`; `--chr`, `--maf`, `--max-missing`, `--remove-filtered-all`, `--remove-indels`, `--thin`, `--phased`; `--keep`, `--remove`), writes output identical byte for byte to vcftools' at any thread count (`SWIFTLD_THREADS`, default all CPUs) and exits non-zero wherever vcftools fails. The reference binary carried one upstream fix, for the temporary-file name buffer (Section 2.5), without which every LD run aborts on Ubuntu. SwiftLD vendors libdeflate 1.26 with three small patches. I developed SwiftLD with correctness-gated coding agents (Claude Opus 5.5, Anthropic), following SwiftNJ (Christensen 2026): every code change had to pass the correctness test in Section 4.2.

4.2 Correctness testing

Each test ran SwiftLD and the live vcftools binary on the same random input: a small adversarial VCF file with 1 to 65 samples and up to 120 records, mixing phased, unphased, haploid and triploid calls, missing data, multi-allelic, indel and `ALT=.` records, unsorted positions, interleaved chromosomes and gzip input, run with random options and 1 to 7 threads. The output had to match byte for byte, and a failing vcftools run had to be matched by a failing SwiftLD run. Each check ran 300 such tests from a fresh seed and compared the output on each of the six development sets with the checksum recorded for vcftools. The five held-out sets were checked six times and never used to choose between versions.

4.3 Data

The real data are five regions of the 1000 Genomes Phase 3 release of chromosome 22 (GRCh37; 2,504 samples phased with SHAPEIT2; 1000 Genomes Project Consortium 2015), pinned by sha256, with every record kept. The three development regions are EUR (503 samples, 1 Mb), all populations (2,504 samples, 0.25 Mb) and AFR (661 samples, 0.5 Mb, phase removed for `--geno-r2`); the two held-out regions are EAS (504 samples, 0.5 Mb) and SAS (489 samples, 0.3 Mb, phase removed). Three synthetic development sets and three synthetic held-out sets from a haplotype-mosaic generator add all-pairs output, missing and unphased calls, extra FORMAT fields, multi-allelic records, several chromosomes and 1,500 samples (Supplementary Table S1).

4.4 Timing

Times are wall times of the whole command, including reading, decompressing, computing and writing the output to a file. Headline timings come from a cloud VM with 2 vCPUs (Intel Xeon at 2.1 GHz, AVX-512, 7.8 GB RAM, g++ 13.3.0). vcftools ran three times per set and the best time counts. SwiftLD's two-thread times are the median of three evaluation runs on the final code, each per-set time the best of up to three runs; the held-out sets and the one-thread times come from one evaluation run each. The headline is the median of the three runs' geometric means over the development regions, and every speed-up is relative to vcftools on the same machine. The whole protocol was repeated on a second VM of the same type, with vcftools re-timed there, and on a laptop (AMD Ryzen 5 5600H, Zen 3, 6 cores and 12 threads, AVX2 without AVX-512, WSL2 Ubuntu 24.04, gcc 13.3) at 12 threads.

4.5 Baselines

PLINK 2.0 and PLINK 1.9 (Chang et al. 2015; plink-ng commit 03bc669) and scikit-allel 1.3.13 (Miles et al. 2024) ran on the five 1000 Genomes regions on the same VM, with windows and allele frequency thresholds matching vcftools': PLINK 2 `--r2-phased` or `--r2-unphased` and PLINK 1.9 `--r2`, both at two threads after importing the VCF, and scikit-allel `rogers_huff_r` (Rogers and Huff 2009) on haplotypes or dosages. One script timed every tool, SwiftLD included, as the best of two runs for runs under 5 s; it timed SwiftLD at 0.025 to 0.078 s, against 0.020 to 0.049 s in Table 1, and every ratio to another tool uses SwiftLD's time from that script. Agreement of r^2 with vcftools was measured on the pairs both tools reported (Supplementary Section S2).

Data and code availability

The code is available at https://github.com/joe-b-christensen/swiftld and archived on Zenodo (DOI: 10.5281/zenodo.23562615). SwiftLD is free software under the GNU Lesser General Public License v3.0, the licence of vcftools, from whose behaviour it derives; it vendors libdeflate 1.26 (MIT licence) with three small patches. The repository contains the source code, its build script, the scripts that check the output against vcftools and time both programs (prepare.py, evaluate.py, gen_data.py) and the five 1000 Genomes inputs.

The synthetic datasets are regenerated deterministically by gen_data.py. The real inputs are subsets of the 1000 Genomes Project Phase 3 release (1000 Genomes Project Consortium 2015), file ALL.chr22.phase3_shapeit2_mvncall_integrated_v5b.20130502.genotypes.vcf.gz and sample panel integrated_call_samples_v3.20130502.ALL.panel from http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/, retrieved on 11 October 2026. These data are openly available without restriction under the data reuse policy of the International Genome Sample Resource (https://www.internationalgenome.org). The five subsets (4.7 MB) are in the repository's `data/` folder with their sha256 checksums, together with the scripts that recreate them from the release file.

Competing interests

None declared.

Funding

None.

References

1000 Genomes Project Consortium (2015). A global reference for human genetic variation. Nature 526:68–74. doi:10.1038/nature15393

Biggers E (2026). libdeflate, version 1.26. https://github.com/ebiggers/libdeflate

Chang CC, Chow CC, Tellier LCAM, Vattikuti S, Purcell SM, Lee JJ (2015). Second-generation PLINK: rising to the challenge of larger and richer datasets. GigaScience 4:7. doi:10.1186/s13742-015-0047-8

Christensen J (2026). SwiftNJ: Fast Exact Neighbour Joining via Correctness-Gated Coding Agents. bioRxiv preprint. doi:10.64898/2026.05.28.728410

Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST, McVean G, Durbin R, 1000 Genomes Project Analysis Group (2011). The variant call format and VCFtools. Bioinformatics 27:2156–2158. doi:10.1093/bioinformatics/btr330

Hill WG, Robertson A (1968). Linkage disequilibrium in finite populations. Theoretical and Applied Genetics 38:226–231. doi:10.1007/BF01245622

Lewontin RC (1964). The interaction of selection and linkage. I. General considerations; heterotic models. Genetics 49:49–67. doi:10.1093/genetics/49.1.49

Miles A, et al. (2024). scikit-allel, version 1.3.13. Zenodo. doi:10.5281/zenodo.13772087

Rogers AR, Huff C (2009). Linkage disequilibrium between loci with unknown phase. Genetics 182:839–844. doi:10.1534/genetics.108.093153

Slatkin M (2008). Linkage disequilibrium: understanding the evolutionary past and mapping the medical future. Nature Reviews Genetics 9:477–485. doi:10.1038/nrg2361