



Sequence analysis

# Insane in the vembrane: filtering and transforming VCF/BCF files

Till Hartmann <sup>1\*</sup>, Christopher Schröder<sup>2</sup>, Elias Kuthe<sup>3</sup>, David Lähnemann <sup>1,4†</sup> and Johannes Köster<sup>1,5†</sup>

<sup>1</sup>Algorithms for Reproducible Bioinformatics, University Hospital Essen, University of Duisburg-Essen, Essen 45147, Germany, <sup>2</sup>Genome Informatics, Institute of Human Genetics, University Hospital Essen, University of Duisburg-Essen, Essen 45147, Germany, <sup>3</sup>Computer Science XI, TU Dortmund University, Dortmund 44227, Germany, <sup>4</sup>Department of Medical Oncology, West German Cancer Center, University Hospital Essen, University of Duisburg-Essen, Essen 45147, Germany and <sup>5</sup>Department of Medical Oncology, Dana-Farber Cancer Institute, Harvard Medical School, Boston, MA 02215, USA

\*To whom correspondence should be addressed.

†The authors wish it to be known that, in their opinion, the last two authors should be regarded as Joint Last Authors.

Associate Editor: Janet Kelso

Received on August 22, 2022; revised on November 15, 2022; editorial decision on December 12, 2022; accepted on December 14, 2022

## Abstract

**Summary:** We present *vembrane* as a command line variant call format (VCF)/binary call format (BCF) filtering tool that consolidates and extends the filtering functionality of previous software to meet any imaginable filtering use case. *Vembrane* exposes the VCF/BCF file type specification and its unofficial extensions by the annotation tools *VEP* and *SnpEff* as Python data structures. *vembrane filter* enables filtration by Python expressions, requiring only basic knowledge of the Python programming language. *vembrane table* allows users to generate tables from subsets of annotations or functions thereof. Finally, it is fast, by using *pysam* and relying on lazy evaluation.

**Availability and implementation:** Source code and installation instructions are available at [github.com/vembrane/vembrane](https://github.com/vembrane/vembrane) (doi: 10.5281/zenodo.7003981).

**Contact:** [till.hartmann@tu-dortmund.de](mailto:till.hartmann@tu-dortmund.de)

**Supplementary information:** [Supplementary data](#) are available at *Bioinformatics* online.

## 1 Introduction

Identifying variation from DNA- or RNA-sequencing data and determining its effect on phenotypes are at the heart of a wide range of biological and medical research efforts. Initial bioinformatics processing of such sequencing data obtains thousands to millions of individual differences between one or more biological samples and their reference genome. These variants are annotated with data properties and known or predicted phenotypic effects and usually stored in the variant call format (VCF) or its binary equivalent (BCF) (Danecek *et al.*, 2011). This annotation information can then be used to filter down to a set of interesting candidate variants, for example, those known to be drug targets in a specific disease.

Here, we present *vembrane*, a new filtering tool for all versions of the VCF and BCF formats. *vembrane* consolidates and extends the functionality of previously available tools and uses standard Python syntax, while achieving very good processing speed. The direct use of Python syntax enables flexible and powerful expressions (Fig. 1) and obviates the need to adapt to a new syntax for users already familiar with Python. It supports both *SnpEff* (Cingolani *et al.*, 2012; Ruden *et al.*, 2012) and *VEP* (McLaren *et al.*, 2016)

annotations out of the box and has an extensible design which allows easy integration of new annotation sources. To our knowledge, it is the only variant filtering tool that can handle groups of breakend events that represent structural variants. It consists of three subcommands for processing VCF records: *filter* for filtering, *table* for converting into a tabular format and *annotate* for adding additional annotations based on genomic ranges.

## 2 Materials and methods

### 2.1 Implementation

For each record, *vembrane* evaluates a given Python expression which has access to all fields defined in the VCF specification as local variables (CHROM, POS, ID, REF, ALT, QUAL, FILTER, INFO, FORMAT), typed according to the VCF file's header and parsed via *pysam*. Variant annotation is available via the ANN dictionary. Annotations from *SnpEff* and *VEP* have custom parsers in *vembrane*, making them easy and safe to use in filter expressions. For VCF records with multiple ANN annotation fields (for multiple affected transcripts), *vembrane* only keeps ANN entries that match

```
vembrane filter '(CHROM == "chr2"
and (QUAL >= 30 or ID in AUX["known"])
and not {"pathogenic", "drug_response"}.isdisjoint(ANN["CLIN_SIG"])
and sum(without_na(FORMAT["DP"])[s] for s in SAMPLES if is_hom(s)) > 10
)' input.bcf --aux known ids.txt > output.vcf
```

Fig. 1. Example invocation of *vembrane filter*. The file *ids.txt* contains one ID per line and is parsed as a set. In plain English, the filter expression translates to ‘keep all records from chromosome 2 where the quality is at least 30 or the ID is in the set of known IDs, and where at least “pathogenic” or “drug\_response” is part of the clinical significance annotations, and where the sum of read depths across all samples that report a homozygous genotype is at least 10’

the expression. The respective record is only kept if at least one ANN passes the filter expression [an implicit any, i.e. conceptually: any(evaluate(expr, ann) for ann in annotations), see [Supplementary Fig. S2](#)].

Records with multiple alternative alleles may have completely unrelated annotations. This both complicates filter expressions and interpretation of variants. Thus, *vembrane* only accepts files whose records have been split such that each alternative allele has its own record. This can, for example, be achieved by normalizing the input with *bcftools norm -N -m -any*; for consistent results, this is best done before annotation.

To our knowledge, *vembrane* is the first tool to comprehensively handle breakend variants (BNDs): BNDs are a way of encoding structural variants by grouping two or more genomic breakpoints into a joint structural variant *event*. As variant files are usually sorted by chromosomal position, BND records from the same event can occur in distant parts of the file. Thus, even if the event it belongs to is known for each BND at the time of reading it, the total number of BNDs (and all associated annotations) for a specific event remains unknown until reaching the end of the file. *vembrane* thus needs to ensure that each event is removed or kept *as a whole*. While non-BND variants are yielded instantly during iteration, BND processing is deferred until sufficient information is available—this is the case as soon as at least one BND of an event passes the filter expression.

## 2.2 Comparison to other tools

Various tools exist for filtering VCF records using conditional expressions over their fields. They vary greatly in the scope of their functionality ([Table 1](#)). For example, the *SnpEff* and *VEP* annotation suites have their own filtering tools, *SnpSift* and *filter\_vep*. Both use custom syntax, special handling of only their own annotations and neither supports the BCF format. Additionally, *filter\_vep* is several orders of magnitude slower than the other tools ([Supplementary Fig. S1](#)). The *bcftools* suite also developed its own expression syntax and supports *VEP* annotations by explicitly activating a dedicated plugin. *bio-vcf* ([Garrison et al., 2021](#)) defines its own domain-specific language for processing VCF files, is multi-threaded by default, but has neither BCF support nor built-in support for annotations. *slivar* ([Pedersen et al., 2021](#)) is geared more toward trio/pedigree filter scenarios, but has some support for specific parts of *SnpEff*, *VEP* and *bcftools* annotations such as Consequence. The only other tool that does not define its own syntax is *VcfFilterJdk* ([Lindenbaum and Redon, 2018](#)), which uses Java expressions for filtering and in principle supports both *VEP* and *SnpEff* (only supports obsolete annotation format from the ‘EFF’ tag) annotations. However, at the time of writing, it produced incorrect VCF v4.2 files. A detailed comparison of specific syntactic capabilities of the different tools, as well as a performance benchmark, can be found in the [supplementary Material](#).

**Table 1.** Comparing different tools/properties, see [Supplementary Material](#) for details.

Tool	Syntax	Annotation	I/O formats	Breakends	Speed
<i>bcftools</i>	Custom	<i>VEP</i> <sup>a</sup>	VCF, BCF	No	+++
<i>bio-vcf</i>	Ruby/custom <sup>b</sup>	-	VCF	No	-
<i>filter_vep</i>	Custom	<i>VEP</i>	VCF	No	-
<i>slivar</i>	Js/custom <sup>b</sup>	Custom <sup>c</sup>	VCF, BCF	No	-
<i>SnpSift</i>	Custom	<i>SnpEff</i>	VCF	No	+
<i>VcfFilterJdk</i>	Java	<i>VEP</i> , <i>SnpEff</i> <sup>d</sup>	VCF, BCF <sup>e</sup>	No	of
<i>vembrane</i>	Python	<i>VEP</i> , <i>SnpEff</i>	VCF, BCF	Yes	++

<sup>a</sup>Through +split-vep plugin.

<sup>b</sup>Additionally ‘custom’ because some scenarios require complex Command Line Interface option combinations.

<sup>c</sup>Special handling of impact annotations from *bcftools*, *VEP* or *SnpEff*.

<sup>d</sup>EFF only.

<sup>e</sup>VCF < v4.3, BCF < v2.1 only.

<sup>f</sup>Manually estimated performance, since it is not included in the benchmark due to incompatible VCF version support and lack of conda packages.

Symbols used for speed classification range from — (slowest) through ° (average) to +++ (fastest).

## 3 Summary

*vembrane* is a software for efficient filtering of data in VCF and BCF files. It combines the capabilities of existing tools and should work as a replacement to any of them. Thus, users will not have to remember which tool can achieve what, but should be able to perform any filtering task with *vembrane*. Further, *vembrane* allows for filtering via arbitrary Python expressions, meaning that Python users can compose filtering expressions without having to learn custom syntax. In addition, it extends beyond existing functionality in other tools by providing support for breakends. Finally, it also allows formatting VCF files into tables and has basic support for annotating records itself.

## Acknowledgements

We sincerely thank Marcel Bargull, Jan Forster, Felix Mölder and Sven Rahmann for their advice.

## Funding

This work was funded by the German Research Foundation collaborative Research Center 876 [SFB 876], subproject C1 (C1), GRF/DFG project number 124020371.

*Conflict of Interest:* none declared.

## References

- Cingolani, P. et al. (2012) Program for annotating and predicting the effects of single nucleotide polymorphisms. *Fly (Austin)*, 6, 80–92.
- Danecek, P. et al.; 1000 Genomes Project Analysis Group. (2011) The variant call format and VCFtools. *Bioinformatics*, 27, 2156–2158.
- Garrison, E. et al. (2021) Vcfliib and tools for processing the VCF variant call format. [bioRxiv](#).
- Lindenbaum, P. and Redon, R. (2018) Bioalcaide, samjs and vcfilterjs: object-oriented formatters and filters for bioinformatics files. *Bioinformatics*, 34, 1224–1225.
- McLaren, W. et al. (2016) The ensembl variant effect predictor. *Genome Biol.*, 17, 1–14.
- Pedersen, B.S. et al. (2021) Effective variant filtering and expected candidate variant yield in studies of rare human disease. *NPJ Genom. Med.*, 6, 1–8.
- Ruden, D. et al. (2012) Using *Drosophila melanogaster* as a model for genotoxic chemical mutational studies with a new program, SnpSift. *Front. Genet.*, 3, 35.