pgen-rs: Efficient and User-friendly Genomic Data Wrangling Aided by LLMs

Ilana Shapiro
UC San Diego
La Jolla, USA

Cole Kurashige
UC San Diego
La Jolla, USA

Savitha Ravi
UC San Diego
La Jolla, USA

ACM Reference Format:
Ilana Shapiro, Cole Kurashige, and Savitha Ravi. 2018. pgen-rs: Efficient and User-friendly Genomic Data Wrangling Aided by LLMs. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In order to perform analyses on genomic data, bioinformaticians often have to pre-process it to meet the formats of their tools or requirements of their studies, such as changing file formats or filtering to a specific subpopulation (examples from Tutorial 2 in [4]). This data wrangling is not only (1) cumbersome because it is mostly done via ad-hoc shell scripts or CLI tools, but also (2) slow due to limitations in the file formats for storing the data itself.

To address theses issues, we developed PGEN-RS. PGEN-RS allows users to (1) write their data wrangling requirements in natural language, which is then converted to an executable DSL and actualized with (2) a Rust-based high-performance genomic data processor enabled by the PLINK file format [2].

1.1 Background

Bioinformaticians have expertise in biology and data analysis. Many conduct research by analyzing population genomes using data science techniques, for example by Genome-Wide Association Studies, which attempt to associate complex traits such as disease proclivity or eye color to particular parts of the genome [4]. The prevailing format for performing these analyses is the variant call format (VCF) first developed for the 1000Genomes project [3]. Existing tools for wrangling VCFs such as `bcftools` [1] are complex many-purpose CLI tools that are not user friendly (Problem (1)).

Improvements in next-generation sequencing (NGS) technologies for DNA have enabled many small bioinformatics laboratories to sequence and analyze entire genomes [5]. These sequencing files are much larger than before, with sizes of 50-100 gigabytes per sample in contrast to the original VCFs for the 1000Genomes project which are orders of magnitude smaller. Because analyses like GWASes which previously ran on VCFs could still take hours or days, new formats such as PLINK [2] had to be developed before it would even be possible to run them on NGS data. Unfortunately, while these new formats enable the actual analyses that bioinformaticians perform to be efficiently run on ever-increasing amounts of data, there has been no efficient equivalent for `bcftools` or similar programs (Problem (2)).

Our tool is exciting because it can help bioinformaticians be more efficient, and by lowering the learning curve to use it, it opens the door to novices.

1.2 Additional Background: PLINK vs VCF for Data Wrangling

Problem (2) relates to the efficiency of data wrangling operations (e.g. filtering or querying) on different genomic data formats. In this section we want to give an intuition for *why* a tool like `bcftools` is held back by the file format it operates on, and how developing a tool for the PLINK format enables it to be more efficient.

An important part of data wrangling is identifying which data to keep and which to discard. The data stored in VCF and PLINK files can be largely divided into two categories: *genotypes* and *metadata*.

1.2.1 Genotypes. Large parts of the human genome are the same. A genotype can be thought of as summarizing what a person's DNA is at each of the varying loci in the genome; this is much more compact than writing out the entire DNA sequence. A genomic data file such as a VCF can be thought of as a matrix where each row corresponds to a locus of variation (known as a "variant") in the genome and each column corresponds to the genotype of an individual (known as a "sample").

1.2.2 Metadata. Though the vast majority of these files are just the (column-wise) concatenation of many individual genotypes, they also contain metadata. At each variant (i.e. on each row), they record information such as what chromosome and position in the chromosome the variant comes from and the quality of the data available at this variant (sequencing is not always 100% accurate). At each sample (i.e. on each column) they record information such as the sample identifier.

A key takeaway is that *the metadata is at least an order of magnitude smaller than the genotypes.* A VCF containing the genotypes of 1000 samples takes gigabytes, but its metadata takes megabytes. And the metadata *does not scale significantly* as the number of genotypes increases. A large part of identifying which data to keep can be done using only metadata. The inherent limitation of VCF is that metadata is coupled with genotypes, so *in order to filter a VCF, you must read the entire file*, even though just the metadata would suffice. Tools like `bcftools`, which process VCFs, deal with this by building indices over the data, but in our observations these indices are either not helpful or not used for data wrangling.

The PLINK format addresses this by decoupling metadata and genotypes. Unlike a VCF, a PLINK "file" actually consists of three

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

separate files: one file containing *just* the genotypes and two files containing the variant and sample metadata, respectively. This decoupling means that it is possible to first filter the metadata (which is much smaller), take only the genotypes corresponding to the filtered metadata, and then finish filtering these remaining genotypes (if the filtering also depends on the genotypes themselves).

In summary, PLINK filters genomic data much more efficiently than VCF due to its separation of metadata and genotypes.

2 METHODOLOGY

We began our project with the intent of following a participatory design approach, seeking out meetings with students and professors in UCSD's bioinformatics program who perform analyses on genomic data. After confirming Problems (1) and (2) with them, we sought to address these problems by improving upon a prototype for genomic data wrangling developed by some of the authors for a previous class project. This prototype, though efficient because it uses PLINK instead of VCF files and it is written in Rust, did not fully support the data wrangling in the workflows of UCSD bioinformaticians. Thus, to fully address Problem (2), we made several improvements, particularly the ability to mix metadata and genotype queries.

This initial version of PGEN-RS was just a tool for efficiently wrangling genomic data, but this did nothing to address Problem (1): at this stage it was still a CLI application which required its users to learn a domain-specific language (DSL) in order to use it, like they would if they were using `bcftools`. To address this barrier to entry, we developed a natural language interface for PGEN-RS using an LLM (GPT-3.5). Now users need not learn an entire DSL: they can describe their requirements in natural language to produce efficient, executable queries in the DSL. Because the LLM produces the queries as opposed to wrangling the data directly, these queries can then be reused or incorporated into a bioinformatician's end-to-end data analysis workflow.

2.1 Demo

Please find the demo at https://youtu.be/qDJxDcQ1CJA

3 RESULTS

We conducted tests of GPT-3.5's ability to synthesize queries in the PGEN-RS DSL, evaluating the results using a pass@3 metric. The results of the tests are shown in Table 1. Of the 10 natural language prompts, we have

$$\begin{aligned} \text{pass@1} &= 70\% \\ \text{pass@2} &= 70\% \\ \text{pass@3} &= 70\% \end{aligned}$$

More attempts did not improve accuracy, and for some questions, the first attempt was the only one that was correct.

The first four questions tested basic queries involving the variant and sample metadata files: "I want all the variants with a quality value of 100," "I want all the samples with an IID of NA21106." These basic queries required filtering the values of certain columns given by name. The next six queries tested the ability of the LLM to write queries that only used parts of the `INFO` column. GPT was given a list of keys in the `INFO` column and their descriptions. The 5th, 6th, 7th, and 9th natural language queries were specifically phrased to include either the key name or the wording from the descriptions. The most common mistake made was not including `INFO[<keyname>]`, followed by incorrect use of escape quotes. For the last question, "I want all the variants with an allele frequency less than 0.6 in the European population," the given files did not mention that `EUR_AF` meant *European* allele frequency, but GPT was able to infer this meaning.

Finally, we also successfully used our tool to query and filter a test pgen file. The queries were run over both sample and variant metadata, and the filter included a predicate over both sample and variant metadata. The result of applying the filter was successfully compiled to VCF. Please see the video demo for the specific results of this experiment.

4 REFLECTIONS

4.1 Outcomes

Throughout this project, we immersed ourselves in the minds of bioinformaticians in a combination of in-person interviews and online research. We gained significant domain knowledge, such as how they wrangle data on a daily basis, and the challenges that they face when doing so. We now have an understanding of how genomic data is structured, and what gaps there are in the tooling currently available to bioinformaticians working with this data. This formed the basis for PGEN-RS. Finally, we also learned how to incorporate LLMs into our tooling in order to enhance the user experience, and then how to combine this high-level interface with the high-performance Rust code we wrote on the backend. This gave us a nice balance of usability and efficiency.

We feel that our frontend came out well in integrating with the LLM, and our backend indeed executes extremely efficiently over very large datasets. We also feel that we were able to successfully incorporate domain specific needs we gathered from bioinformatiticans in our limited interviews. However, we do feel that our UI could be improved by eliminating the need for a separate terminal to read results. We also struggled to recruit bioinformaticians to interview, and would like to interview more experts in the future to enhance our outcomes.

4.2 Future Work

Our next steps for this project are to meet with some bioinformaticians again to see what they think of PGEN-RS and have them try out the tool in their daily work. As part of the participatory design process, we would iterate on this design according to their comments. One avenue for future work is to to provide support for more complex predicates to match existing tools like BCFTOOLS, and also eliminate the need for multiple UIs by rendering the query results in our custom frontend, instead of terminal.

Furthermore, we hope to improve the executable DSL syntax so it is easier for the LLM to write queries in it. One hypothesis we have is that making the DSL's syntax look more like Python or Javascript may help it produce fewer syntax errors since it has seen more examples of syntactically-correct Python code.

5 ARTIFACT

Please find the artifact at https://github.com/ilanashapiro/pgen-rs.

Natural Language Query	Answers	Score
I want all the variants with a quality value of 100	QUAL == "100" QUAL == "100" QUAL == "100"	3/3
I want all the variants with a reference allele of C or A	REF=="C" || REF=="A" REF == "C" || REF == "A" REF == "C" || REF == "A"	3/3
I want all the variants with an alternative allele of T and a passing value for the filter	ALT=="T" && FILTER=="PASS" ‘ALT=="T" && FILTER=="PASS"‘ ALT == "T" && FILTER == "PASS"	2/3
I want all the samples with an IID of NA21106	IID == "NA21106" “‘ \n IID == "NA21106" \n “‘ “‘ \n IID == "NA21106" \n “‘	1/3
I want all the variants with an allele count of 1852	INFO[AC] == "1852" INFO[AC] == "1852" INFO[AC] == "1852"	3/3
I want all the variants with a variant type representing a single nucleotide polymorphism	SVTYPE == "SNP" VT=="SNP" VT=="SNP"	0/3
I want all the variants with an EAS allele frequency of less than 0.1	EAS_AF <"0.1" EAS_AF <"0.1" EAS_AF<"0.1"	0/3
I want all the variants that have 2504 samples with data	NS == "2504" NS == "2504" NS == "2504"	0/3
I want all the variants with a total read depth less than 20070 and an allele count greater than 2000	INFO[DP] < "20070" && INFO[AC] > "2000" “‘ \n INFO[DP] < "20070" && INFO[AC] > "2000" \n “‘ INFO[DP] < "20070" && INFO[AC] > "2000"	2/3
I want all the variants with an allele frequency less than 0.6 in the European population	INFO[EUR_AF] < "0.6" EUR_AF < "0.6" EUR_AF < "0.6"	1/3

Table 1: Table of results for natural language queries

REFERENCES

[1] [n. d.]. Bcftools. https://samtools.github.io/bcftools/. Accessed: 2024-06-05.
[2] Christopher C Chang, Carson C Chow, Laurent CAM Tellier, Shashaank Vattikuti, Shaun M Purcell, and James J Lee. 2015. Second-generation PLINK: rising to the challenge of larger and richer datasets. *Gigascience* 4, 1 (2015), s13742–015.
[3] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A Albers, Eric Banks, Mark A DePristo, Robert E Handsaker, Gerton Lunter, Gabor T Marth, Stephen T Sherry, et al. 2011. The variant call format and VCFtools. *Bioinformatics* 27, 15 (2011), 2156–2158.
[4] Melissa Gymrek. [n. d.]. Personal Genomics for Bioinformaticians Chapter 7: GWAS for complex traits. https://gymrek-lab.github.io/personal-genomics-textbook/complextraits/gwas.html. Accessed: 2024-06-05.
[5] Erwin L. van Dijk, Hélène Auger, Yan Jaszczyszyn, and Claude Thermes. 2014. Ten years of next-generation sequencing technology. *Trends in Genetics* 30, 9 (2014), 418–426. https://doi.org/10.1016/j.tig.2014.07.001