

1 **Genomic Analyzer: A Scalable and High-** 2 **Performance Object-Oriented Application for SNP** 3 **Analysis and Genetic Association Studies** 4

5 *Bioinformatics flourished in medical fields in the last decade by making use of the*
6 *exponential growth in computer science expertise. Single Nucleotide Polymorphism*
7 *(SNP) is the most common type of genetic variation among people, where a single*
8 *building block of DNA is different from the reference sequence. These variations are*
9 *found in at least 1% of the population and can serve as biological markers. SNPs are*
10 *required as the main input for Genome-Wide Association Study (GWAS) where the*
11 *correlation between SNPs and phenotypes is demonstrated. This correlation is*
12 *important for many medical fields such as disease susceptibility, disease progression,*
13 *survival period, and drug response. The increasing availability of genomic data has*
14 *created a demand for tools capable of efficiently processing and analyzing single*
15 *nucleotide polymorphism (SNP) datasets. Existing tools often rely on command-line*
16 *interfaces and fragmented workflows, creating barriers for accessibility and*
17 *reproducibility. This paper presents the Genomic Analyzer, a modular graphical*
18 *desktop application designed to streamline preprocessing, quality control, and analysis*
19 *of SNP datasets. The system introduces pipeline-based architecture with real-time*
20 *dataset previews and a JSON-based run manifest for reproducibility. The proposed*
21 *approach improves usability while maintaining extensibility for future integration of*
22 *machine learning models.*

23
24 **Keywords:** *Genomic Analyzer, SNP Analysis, Genome-Wide Association Study (GWAS),*
25 *Software Engineering, Object-Oriented Application, Scalability, Preprocessing Pipelines,*
26 *Reproducibility, and High-Performance.*

27 28 29 **Introduction** 30

31 SNP datasets are widely used in genome-wide association studies (GWAS),
32 yet they often originate from heterogeneous sources with inconsistent formats.
33 This variability introduces significant preprocessing overhead and increases the
34 likelihood of errors. Existing tools, such as PLINK and SNPTest, require
35 command-line interaction (CLI) which presents a steep learning curve and limits
36 accessibility for new researchers.

37 CLI tools such as PLINK produce dense textual outputs that require
38 significant domain knowledge to interpret, limiting accessibility for new users.
39 Even after initial familiarity, the interface remains difficult to interpret,
40 reproduce, and analyze.

41 Additionally, current workflows typically require chaining multiple tools
42 together, including preprocessing utilities and analysis engines (Chen et al.,
43 2009), often without a unified system for tracking transformations. This lack of
44 integration reduces reproducibility and complicates experimental validation. As
45 genomic datasets continue to grow in size and complexity, there is a need for
46 systems that simplify preprocessing while maintaining analytical rigor. The
47 objective of this work is to develop a modular, extensible genomic analysis

1 system that eliminates manual preprocessing steps, improves reproducibility,
2 and provides an intuitive graphical interface for users.

3 4 5 **Literature Review**

6
7 Existing GWAS tools primarily fall into two categories: command-line tools
8 and graphical tools. (Chen et al., 2009; Balagué-Dobón et al., 2022). Command-
9 line tools such as PLINK and SNPTest provide powerful functionality but
10 require significant expertise to operate effectively. Their workflows are often
11 static and require manual re-entry of parameters for iterative experimentation.

12 Graphical tools such as SNPStats attempt to improve usability but introduce
13 other limitations, including restricted execution time and limited extensibility.
14 These systems do not support modular pipeline construction, making it difficult
15 to adapt workflows to new requirements.

16 The absence of unified, extensible architecture highlights a gap in current
17 bioinformatics tooling. There is a need for systems that combine the power of
18 existing analysis engines with the flexibility of modular design and the
19 accessibility of graphical interfaces.

20 21 *Genome-Wide Association Studies and Tooling Requirements*

22
23 Genome-wide association studies (GWAS) are a fundamental method in
24 computational biology used to identify associations between genetic variants and
25 observable traits or diseases. These studies rely on large datasets containing
26 single nucleotide polymorphisms (SNPs), which represent small variations in
27 DNA sequences across individuals. By analyzing these variations across
28 populations, researchers can detect statistically significant correlations between
29 specific genetic markers and phenotypic outcomes.

30 The computational requirements of GWAS workflows are substantial. Raw
31 genomic datasets are often heterogeneous, containing inconsistent genotype
32 encodings, varying delimiters, and multiple representations of missing data, as
33 commonly observed in genomic datasets (Balagué-Dobón et al., 2022). Before
34 statistical analysis can be performed, these datasets must be normalized into a
35 consistent internal format. Additionally, quality control (QC) steps are required
36 to ensure data integrity in GWAS workflows (Cordell and Clayton, 2005),
37 including the computation of metrics such as minor allele frequency (MAF),
38 Hardy–Weinberg equilibrium (HWE), and missingness rates. These
39 preprocessing and QC stages are essential for producing valid and reproducible
40 results.

41 As a result, GWAS tools must support complex data transformation
42 pipelines, statistical analysis, and reproducibility tracking. However, these
43 requirements introduce significant usability and workflow challenges. Tools
44 must balance computational rigor with accessibility, while also providing
45 mechanisms for validating intermediate transformations and ensuring that
46 analysis steps can be reproduced across different environments and timeframes.

1 The design of effective GWAS tooling therefore lies at the intersection of
2 domain-specific requirements and general software engineering principles.

3 4 Minor-Allele Frequency

5 Minor allele frequency (MAF) is a fundamental metric in genome-wide
6 association studies that measures the frequency of the less common allele at a
7 given genetic locus within a population (Zheng et al., 2012). It is calculated as
8 the proportion of occurrences of the minor allele relative to the total number of
9 observed alleles across all samples.

10 MAF is used as a quality control filter to remove variants that are too rare
11 to provide statistically meaningful results. Extremely low-frequency alleles may
12 arise from sequencing errors or insufficient sample representation and including
13 them in analysis can introduce noise or instability in statistical models. As a
14 result, GWAS workflows commonly apply a minimum MAF threshold to
15 exclude such variants before downstream analysis.

16 17 Hardy-Weinberg Equilibrium

18 Hardy-Weinberg equilibrium (HWE) is a principle that describes the
19 expected distribution of genotype frequencies in a population under ideal
20 conditions, assuming no evolutionary pressures such as selection, mutation, or
21 migration. In GWAS, deviations from this expected distribution are used as an
22 indicator of potential data quality issues.

23 Statistical tests, typically based on chi-square or exact methods, are used to
24 compute a p-value representing the likelihood that observed genotype
25 frequencies conform to HWE expectations. Variants that significantly deviate
26 from equilibrium may indicate genotyping errors, population stratification, or
27 data inconsistencies, and are therefore often excluded as part of quality control
28 preprocessing (Cordell and Clayton, 2005).

29 30 Missingness Rates

31 Missingness rate refers to the proportion of missing genotype data within a
32 dataset, measured either per SNP or per sample. It quantifies the extent to which
33 data is incomplete, which can significantly impact the validity of downstream
34 statistical analyses.

35 High missingness rates may indicate poor data quality, errors during data
36 collection, or inconsistencies in dataset formatting. GWAS workflows typically
37 apply threshold-based filters to exclude SNPs or samples with excessive missing
38 values, ensuring that analyses are performed on sufficiently complete and
39 reliable data (Cordell and Clayton, 2005). This metric is particularly important
40 in preprocessing pipelines, where consistent handling of missing values is
41 required to maintain data integrity.

42 43 *PLINK CLI Tool*

44
45 PLINK is a widely used command-line toolset for genome-wide association
46 studies (GWAS) and population-based genetic analysis (Chang et al., 2015). It

1 provides functionality for data management, statistical association testing, and
 2 computation of quality control (QC) metrics such as minor allele frequency
 3 (MAF), Hardy–Weinberg equilibrium (HWE), and missingness rates.

4 How PLINK Works

6 PLINK operates through a command-line interface (CLI), where users
 7 execute analysis steps by specifying parameters and input files directly in
 8 terminal commands. Typical workflows involve multiple sequential commands
 9 for data preprocessing, filtering, QC, and analysis, often requiring intermediate
 10 file generation between steps. Results are returned as text-based logs and output
 11 files, which must be manually interpreted or further processed.

13 **Figure 1.** *Example of PLINK command-line workflow of filtering*

Remove bad SNPs

```
echo "rs00003" > bad.snps
echo "rs00013" >> bad.snps

./plink --file mygene
--exclude bad.snps
--recode
--out cleaned
```

Equivalent command, using filters

```
./plink --file mygene
--hwe 1e-3
--hwe-all
--geno 0.1
--recode
--out cleaned
```

Normally, HWE filters on controls; the --hwe-all flag implies all individuals

14 *Source:* Purcell 2009.

17 PLINK Limitations

18 Despite its computational power, PLINK presents several usability and
 19 workflow limitations. The command-line interface introduces a steep learning
 20 curve, particularly for users without prior experience in terminal-based tools.
 21 This workflow reflects legacy interaction models and lacks support for modern,
 22 interactive pipeline construction. Workflows are largely static and require
 23 manual re-entry of commands for iterative experimentation (see Figure 1),
 24 increasing the likelihood of user error. This issue becomes more pronounced
 25 when specifying extensive parameter lists for SNP identifiers and filtering
 26 operations. Additionally, PLINK does not provide integrated support for tracking
 27 preprocessing steps or configurations, making reproducibility dependent on
 28 external documentation. The absence of real-time feedback further limits the
 29 ability to validate intermediate transformations during pipeline development.

31 *SNPTest CLI Tool*

33 SNPTest is a statistical analysis tool designed for GWAS, with a focus on
 34 testing associations between genetic variants and phenotypic traits. It supports a
 35 range of statistical models, including frequentist and Bayesian approaches, and
 36 is commonly used for analyzing genotype probabilities and imputed data.

1 How SNPTest Works

2 SNPTest is primarily operated through a command-line interface, where
 3 users specify input datasets, statistical models, and parameters via structured
 4 commands. The tool processes genotype and phenotype data to compute
 5 association statistics, producing output files that contain p-values, effect sizes,
 6 and other relevant metrics. Workflows typically involve multiple preprocessing
 7 and configuration steps performed outside the tool, requiring users to manage
 8 data formatting and pipeline sequencing manually.

10 **Figure 2. Example of SNPTest command line execution**

Example 2 - Conditioning on covariates that code for population structure

For association studies it has become popular to use eigenvectors from a PCA analysis to code for unobserved population structure. This is conditioning upon these covariates. Here is an example of conditioning upon the two continuous covariates called `cov3` and `cov4` in the same

```
./snptest \  
-data ./example/cohort1.gen ./example/cohort1.sample ./example/cohort2.gen ./example/cohort2.sample \  
-o ./example/ex.out \  
-frequentist 1 \  
-method score \  
-pheno bin1 \  
-cov_names cov3 cov4
```

Example 3 - Conditioning on SNPs

In regions where an association has been found it is often desirable to carryout a test conditioning upon the most associated SNP to look for This can be carried out in SNPTEST using the `-condition_on` option. A list of SNPs can be specified along with the coding to be applied to the `RSID_10` and `RSID_20`. The SNP `RSID_10` is coded as an additive effect while SNP `RSID_20` is coded as a general effect.

```
./snptest \  
-data ./example/cohort1.gen ./example/cohort1.sample ./example/cohort2.gen ./example/cohort2.sample \  
-o ./example/ex.out \  
-frequentist 1 \  
-method score \  
-pheno bin1 \  
-condition_on RSID_10 add RSID_20 gen
```

11
12 *Source:* Marchini et al. 2010.

14 SNPTest Limitations

15 While SNPTest provides advanced statistical capabilities, it shares many
 16 usability limitations with other command-line tools. The interface requires
 17 familiarity with complex parameter configurations (see Figure 2), making it less
 18 accessible to new users. Additionally, SNPTest does not provide built-in
 19 mechanisms for iterative workflow construction or visualization of intermediate
 20 results, limiting its effectiveness in exploratory analysis. As with similar tools,
 21 reproducibility depends on external tracking of commands and configurations,
 22 increasing the risk of inconsistency across experiments.

24 *SNPStats GUI Tool*

25
26 SNPStats is a web-based graphical tool designed to support genome-wide
 27 association studies by providing statistical analysis and visualization of SNP
 28 data. It offers functionality for computing association metrics, performing
 29 quality control analyses, and generating human-readable outputs such as tables
 30 and graphical summaries in PDF format. Compared to command-line tools,
 31 SNPStats aims to improve accessibility through a graphical user-interface (GUI).

1 How SNPStats Works

2 SNPStats operates through an online interface where users upload datasets,
3 configure preprocessing and analysis parameters, and execute workflows
4 remotely. The system provides a structured sequence of steps for data
5 preparation, statistical analysis, and result generation. Outputs are delivered as
6 formatted reports, typically in PDF form, containing statistical summaries and
7 visualizations.

8
9 **Figure 3.** *SNPStats preprocessing interface*

Step 2: Data preprocessing

In this step you can see some information obtained for each one of the columns of your data matrix. Please check that the type we assume for every one of them is correct. If there is a SNP column which is considered by the software as a covariate please check your data, as there may be a mistake in your data file. The default reference category for each SNPs is the most frequent homozygote, and for the covariates the first in alphabetical order. All SNPs and covariates not explicitly discarded will be included in the analysis. Only one variable of type "response" is allowed.

File name:
Number of processed rows: 706
Number of processed columns: 8
Column headers: Yes

Column	Name	Type	Number of different values	Number of missings	Category order	Include SNP/covariate
1	group	Categorical covariate	2	0	Ca (377) Co (329)	<input type="checkbox"/>
2	sex	Categorical covariate	2	0	Female (306) Male (400)	<input type="checkbox"/>
3	age	Response	61	0	23 (1) 24 (1)	<input checked="" type="checkbox"/>
4	age.cat	Quantitative covariate	4	0		<input checked="" type="checkbox"/>
5	colon.r	Categorical covariate	3	0	No cancer (359) Other cancers (271)	<input type="checkbox"/>
6	snp1	SNP	3	20	C/C (595) C/T (88)	<input checked="" type="checkbox"/>
7	snp2	SNP	3	23	G/G (602) A/G (76)	<input checked="" type="checkbox"/>
8	snp3	SNP	3	29	G/G (291) A/G (305)	<input checked="" type="checkbox"/>

Include all SNPs Include all covariates

<<< Step 1: Data uploading | Step 3: Analysis customization >>>

10
11 *Source:* SNPStats 2026.

12 SNPStats Limitations

13 While SNPStats improves usability compared to CLI tools, it introduces
14 several limitations that restrict its effectiveness for modern workflows.

15 *Preprocess steps and process*

16
17 The system does not persist in preprocessing configurations between runs,
18 requiring users to manually reconstruct analysis steps for each execution, which
19 reduces reproducibility. Additionally, preprocessing and analysis steps cannot be
20 incrementally verified during setup, limiting the ability to validate
21 transformations before full execution.

22 *Platform constraints*

23
24 The platform imposes execution constraints, including a maximum runtime
25 limit (e.g., approximately ten minutes), which is insufficient for larger genomic
26 datasets. Its web-based architecture also requires an active internet connection,
27 which may be undesirable in environments with data privacy concerns or
28 restricted network access.

1 *Outdated application design*

2
3 From a usability perspective, the interface includes outdated interaction
4 patterns, such as manual button-based reordering instead of modern drag-and-
5 drop mechanisms, which reduces efficiency during workflow construction (see
6 Figure 3). Although SNPStats provides structured PDF outputs with visual
7 summaries, these reports are static and not tightly coupled with the underlying
8 configuration, making it difficult to trace results back to specific preprocessing
9 decisions or to reproduce analyses without repeating the full setup process. Over
10 time, the absence of persistent configuration increases the likelihood that
11 workflows cannot be reliably reconstructed.

12 13 SNP Stats Overview

14 Despite these limitations, SNPStats remains one of the more accessible tools
15 in this domain, highlighting both the value of graphical interfaces and the need
16 for more flexible, reproducible, and interactive systems.

17 18 *Comparative Analysis of Existing Tools*

19
20 Existing GWAS tools demonstrate a clear divide between computational
21 capability and usability. Command-line tools such as PLINK and SNPTest
22 provide powerful statistical functionality and flexible parameterization but
23 require users to construct workflows manually through sequences of commands.
24 This approach introduces a steep learning curve and limits the ability to
25 iteratively refine analyses, as intermediate transformations are not easily
26 visualized or validated. Reproducibility in these systems depends heavily on
27 external documentation of commands and configurations, increasing the
28 likelihood of inconsistencies across experiments.

29 Graphical tools such as SNPStats attempt to improve accessibility by
30 providing a user interface and structured workflows. While these systems reduce
31 the barrier to entry, they introduce new limitations, including restricted execution
32 environments, limited scalability, and a lack of persistent workflow
33 configuration. In particular, the absence of reusable preprocessing pipelines and
34 real-time validation mechanisms makes it difficult to ensure correctness during
35 pipeline construction. As a result, users must repeatedly reconstruct workflows
36 and rely on final outputs without the ability to verify intermediate states.

37 Across both categories, a common limitation is the lack of modular,
38 reproducible, and interactive pipeline design. Existing tools do not provide a
39 unified framework for constructing, validating, and persisting data
40 transformation workflows. This gap highlights the need for systems that combine
41 the computational strength of established tools with modern software
42 engineering approaches, including modular architecture, real-time feedback, and
43 configuration-driven reproducibility. These observations motivate the
44 development of the Genomic Analyzer, which aims to address these limitations
45 through a pipeline-based, modular extensible GUI.

46

1 **Methodology**

2

3 *System Architecture*

4

5 The Genomic Analyzer is designed as a modular and extension system that
6 separates data transformation, quality control and analysis into independent
7 components. The architecture emphasizes flexibility and adaptability, enabling the
8 system to support evolving workflows and integration with external tools without
9 requiring changes to the core framework. By decoupling system components through
10 well-defined interfaces, the design promotes maintainability, scalability and easy
11 extension.

12 A central goal of this architecture is to provide a foundation for constructing
13 configurable genomic data preprocessing workflows while remaining agnostic
14 to underlying analysis engines. This allows the system to support a wide range
15 of downstream tools and methodologies without constraining users to a specific
16 execution environment or implementation.

17

18 Modular System Design

19 The system adopts a plug-and-play modular design in which functionality
20 is encapsulated within discrete interchangeable modules. Each module performs
21 a specific transformation or evaluation task and operates independently of other
22 modules, interacting only through standardized input and output interfaces. This
23 design allows new functionality to be introduced by adding modules rather than
24 modifying existing system components, reducing the risk of regression and
25 enabling rapid feature extension.

26 Modules can be composed of flexible workflows tailored to specific datasets
27 and analysis requirements. Because modules are self-contained and loosely
28 coupled they can be reused across different workflows and configurations,
29 supporting both experimentation and reproducibility. This modular approach
30 also facilitates incremental system evolution, as new capabilities can be
31 integrated without disrupting established functionality.

32

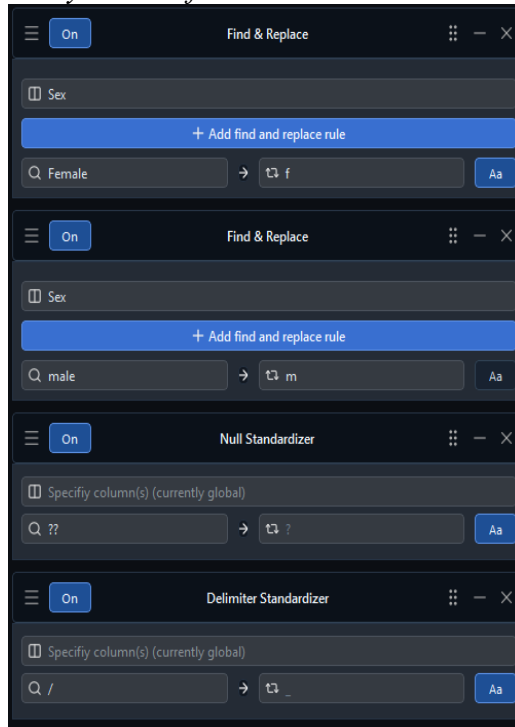
33 Module Abstraction

34 Modules are defined as the fundamental computational units within the
35 system, each representing a discrete operation applied to genomic data. The
36 module abstraction enforces a consistent interface for execution, allowing
37 heterogeneous operations to be composed within a unified workflow. This
38 abstraction enables the system to treat all modules uniformly, regardless of their
39 specific functionality or internal implementation.

40 By isolating functionality within modules, this design allows individual
41 operations to be developed, tested and extended independently. This separation
42 of concerns improves code maintainability and supports clear reasoning about
43 data transformations, as each module is responsible for a well-defined limited
44 scope behavior. This design allows for expandable info-cards at the title-bar or
45 side-menu of each module. This feature allows the interface itself to serve as an
46 educational aid.

1 An arbitrary number of modules of the same type may be instantiated in a
 2 single pipeline, encouraging multiple instances with different parameters or
 3 column scopes.

4
 5 **Figure 4.** *Example of preprocessing modules configured within the Genomic*
 6 *Analyzer interface*



7
 8 *Source:* Created by author.

9 10 *Preprocessing modules*

11
 12 Preprocessing modules are responsible for transforming input datasets into
 13 a consistent internal representation suitable for downstream analysis. Preprocess
 14 modules operate prior to both the quality control pipeline and final analysis.
 15 Operations such as normalization, standardization into a consistent internal
 16 representation, structural transformation, and find and replace commands are
 17 performed here. By isolating these transformations within dedicated modules,
 18 this approach ensures data inconsistencies are resolved in a controlled and
 19 reproducible manner.

20 The design of preprocessing modules prioritizes flexibility, allowing users
 21 to define and compose transformation steps that reflect the specific requirements
 22 of their datasets. This enables the system to accommodate a wide range of input
 23 formats and encoding variations while maintaining a consistent internal
 24 representation.

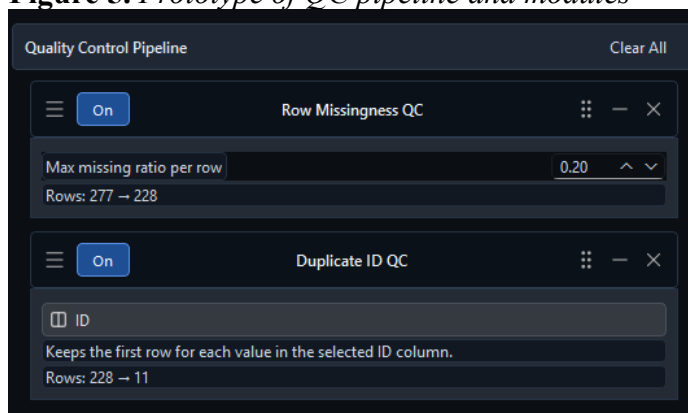
25 The default scope of preprocess modules is a global scope, meaning the
 26 module's changes will affect all columns of the dataset. Preprocess modules have
 27 the option to be scoped to any column(s) (see Figure 4). This further allows

1 greater flexibility, as some transformations would be invalid or unwanted in
2 some columns.

3 4 *Quality control modules*

5
6 Quality control modules operate on the dataset to evaluate data integrity and
7 enforce predefined quality thresholds. These modules compute statistical metrics
8 and apply exclusion criteria consistent with standard GWAS workflows (Cordell
9 and Clayton, 2005) to identify and remove unreliable data points prior to analysis
10 (see Figure 5). Unlike preprocessing modules, which focus on data
11 transformation, quality control modules are concerned with validation and
12 filtering based on domain-specific standards.

13
14 **Figure 5.** *Prototype of QC pipeline and modules*



15
16 *Source:* Created by author.

17 18 *Module composition and reusable components*

19
20 In addition to system-level modularity, modules themselves are constructed
21 from smaller, reusable components that encapsulate common patterns of
22 interaction and configuration. These components represent recurring functional
23 elements, such as column selection, threshold specification, and transformation
24 rule definition, which are shared across multiple module types.

25 This compositional design enables modules to be assembled from
26 standardized building blocks rather than implemented from scratch. For
27 example, threshold-based components can be reused across multiple quality
28 control modules, while transformation components such as find-and-replace
29 logic can be applied across different preprocessing operations. Similarly,
30 column-scoped components provide a consistent mechanism for selecting and
31 applying operations to subsets of data. Reusability extends beyond logical code
32 reuse. It also includes graphical interface components, allowing future modules
33 to be constructed from standardized input components and module-level logic.

34 By promoting reuse at the component level, the system reduces the
35 complexity of developing new modules over time. As the library of reusable
36 components expands, the effort required to implement additional functionality

1 decreases, allowing new modules to be constructed more rapidly and with
2 greater consistency. This results in an extensibility model in which system
3 growth becomes progressively more efficient, rather than increasingly complex.
4

5 *Module discoverability and interaction model*

6

7 Traditional command-line-based genomic analysis tools require users to
8 possess prior knowledge of available functionality, as operations must be
9 explicitly specified through commands and parameters. This creates a
10 discoverability barrier, where users are limited to features they already know or
11 are actively searching for. As a result, workflow construction becomes dependent
12 on external documentation rather than interaction with the system itself.

13 In contrast, the Genomic Analyzer exposes functionality through a module-
14 based graphical interface in which available operations are presented explicitly as
15 composable elements. Modules are selected and arranged within the workflow,
16 allowing users to observe and explore available functionality directly during
17 pipeline construction. This interaction model enables features to be discovered
18 through use, rather than through prior familiarity or external reference.

19 Each module is accompanied by contextual metadata, including descriptions
20 of its purpose, expected inputs, and typical use cases. This information is
21 accessible within the interface, providing immediate guidance without
22 interrupting workflow construction. This approach improves accessibility for
23 new users while also benefiting experienced researchers by reducing the
24 cognitive overhead associated with recalling and configuring complex command
25 sequences. By embedding discoverability within the system's interaction model,
26 the design transforms workflow construction from a purely declarative process
27 into an interactive and informative experience.
28

29 Adapter-Based Engine Integration

30 The Genomic Analyzer employs an adapter-based design to integrate
31 external analysis engines while maintaining a consistent internal interface.
32 Rather than coupling the system directly to any specific tool, each external
33 engine is encapsulated within an adapter that conforms to a shared execution
34 contract. This allows the core system to interact with all engines through a
35 uniform interface, independent of their underlying implementation details.

36 In the current design, integration is initially focused on PLINK as the
37 primary downstream analysis engine, reflecting its widespread use and
38 compatibility with standardized genomic data formats. Support for additional
39 tools, such as SNPTest, is planned as a subsequent extension. Future extensions
40 may include custom analysis engines, including machine learning-based
41 models, without requiring modifications to the existing pipeline or module
42 architecture.

43 By abstracting engine-specific behavior behind adapters, the system
44 achieves engine agnosticism, enabling workflows to be constructed
45 independently of the chosen analysis backend. This allows users to configure
46 and validate preprocessing and quality control pipelines without committing to

1 a specific execution environment. Engine selection becomes a runtime decision,
2 where the same pipeline can be executed against different backends depending
3 on the desired analysis method or output format.

4 This design promotes extensibility and long-term maintainability by
5 isolating external dependencies from the core system. New engines can be
6 integrated by implementing the interface, without impacting existing modules or
7 workflows. As a result, the system supports incremental expansion of analysis
8 capabilities while preserving a stable and consistent user-facing architecture.

9 10 *Pipeline Architecture*

11 The Genomic Analyzer organizes data processing as a sequence of
12 composable operations applied through a structured pipeline. This pipeline
13 serves as the central execution model of the system, enabling users to construct,
14 modify, and evaluate workflows in a controlled and reproducible manner. Each
15 stage in the pipeline represents a deterministic transformation or evaluation step,
16 allowing complex genomic processing tasks to be decomposed into smaller,
17 manageable units.

18 The pipeline is designed to support both interactive development and final
19 execution, providing flexibility during workflow construction while maintaining
20 consistency in analysis. To achieve this, the system defines a unified execution
21 model with distinct operational modes and supporting mechanisms for efficient
22 iteration and validation.

23 24 Pipeline Execution Model

25 The pipeline execution model defines how data flows through modules and
26 how transformations are applied during different stages of workflow
27 development. It distinguishes between interactive execution, where users
28 iteratively refine pipeline behavior, and final execution, where the pipeline is
29 applied to the full dataset. This separation allows the system to optimize for
30 responsiveness during development while preserving correctness and
31 reproducibility during analysis.

32 Execution is structured to ensure that each module operates on the output of
33 its predecessor, forming a deterministic chain of transformations. The system
34 provides mechanisms to control how and when these transformations are
35 evaluated, enabling efficient updates and meaningful inspection of intermediate
36 results.

37 38 *Preview mode (interactive execution)*

39
40 In existing genomic analysis workflows, preprocessing and quality control
41 pipelines are often constructed without immediate validation, increasing the
42 likelihood of undetected errors or malformed data. If such issues are not
43 identified early, they may propagate through the pipeline and result in incorrect
44 downstream analysis. Because full dataset execution can be computationally
45 expensive, these errors are often discovered only after significant time has been

1 spent processing invalid configurations, leading to inefficient and potentially
 2 misleading results.

3

4 **Figure 6.** *Real-time dataset preview during pipeline construction*

Preview									
ID	Affection	Sex	DRB1_1	DRB1_2	SENum	SEStatus	AntiCCP	RFUW	rs10439884
D0024949	no	female	101	401	SS	yes	??	??	a/g
D0024302	??	Male	??	7	NN	yes	13	23	??/c
D0023151	no	??	101	??	SN	??	??	??	t/g
D0022042	??	??	102	2	NN	no	12	??	g/g
D0024949	no	female	??	??	SS	yes	??	??	a/g
D0021275	??	Female	??	7	NN	yes	??	34	a/??
D0021163	no	??	??	403	??	??	32	45	??/c
D0020795	??	??	??	3	NN	no	??	??	t/g
6045201	??	female	101	??	SN	yes	31	14	a/g
D0023027	no	Male	101	??	??	??	??	??	c/t
1015200	??	??	101	??	SN	yes	11	45	??/??
D0015941	yes	??	102	??	NN	??	??	??	a/g
D0016405	??	female	??	7	??	yes	??	??	??/??
D0024949	no	female	??	401	SS	??	??	??	a/g

5

6 *Source:* Created by author.

7

8 Preview mode addresses this limitation by enabling real-time previews of
 9 pipeline behavior during construction via a live dataset. By executing
 10 transformations on a controlled subset of the dataset, the system allows users to
 11 observe the effects of preprocessing and quality control steps as they are defined.
 12 This immediate feedback loop reduces the risk of propagating errors and allows
 13 issues to be identified and corrected before full analysis is performed, as well as
 14 facilitates visualization of the effects of each module (see Figure 6).

15 This approach transforms pipeline construction from a deferred validation
 16 process into an interactive and iterative workflow, where correctness can be
 17 established incrementally. As a result, preview mode improves both the
 18 efficiency and reliability of genomic data processing by ensuring that pipelines
 19 are verified prior to large-scale execution.

20

21 Subset based execution

22 In preview mode, the pipeline operates on a reduced subset of the full dataset
 23 with a limited row count specifiable by the user. This approach significantly
 24 reduces computational overhead, enabling near-instantaneous feedback during
 25 pipeline modification. By constraining execution to a representative sample, the
 26 system allows users to validate transformations without incurring the cost of full
 27 dataset processing.

28

29 The subset is selected in a deterministic manner to ensure consistency across
 30 executions, allowing users to reliably compare results as the pipeline evolves.
 31 This strategy balances performance with fidelity, providing sufficient data to
 observe transformation effects while maintaining interactive responsiveness.

1 Incremental re-computation strategy

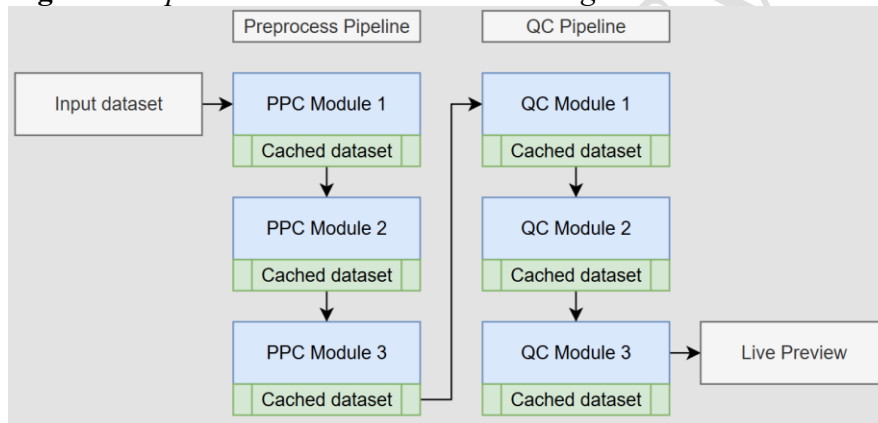
2 To further optimize interactive performance, the system employs an
 3 incremental re-computation strategy in which only affected portions of the
 4 pipeline are re-executed when changes are made. When a module is modified,
 5 all preceding modules remain unchanged, and only the modified module and its
 6 successors are recomputed.

7 This approach avoids redundant computation and ensures that pipeline
 8 updates scale efficiently with the number of modules. By limiting recomputation
 9 to the minimal necessary scope, the system maintains responsiveness even as
 10 pipeline complexity increases.

11 Intermediate state inspection

12 The pipeline execution model supports inspection of intermediate states
 13 between modules, allowing users to observe the output of each transformation
 14 step. This capability provides transparency into pipeline behavior and enables
 15 validation of individual operations prior to full execution.

16 **Figure 7. Pipeline structure and data caching schema**



17
 18
 19
 20 *Source:* Created by author.

21
 22 Each module produces a deterministic intermediate dataset that can be
 23 examined independently, supporting step-by-step visual verification of
 24 transformations. This design reduces the likelihood of compounding errors and
 25 allows issues to be identified and corrected early in the pipeline construction
 26 process. The dataset is cached locally at each step in both pipelines, preventing
 27 a breaking change from corrupting previous steps, thus freeing users from
 28 runtime destructive changes (see Figure 7).

29 *Analysis mode (final execution)*

30
 31
 32 Once a pipeline has been fully constructed and validated by the user, the
 33 user initiates analysis using the configured pipeline. After the analysis is set up,
 34 the complete dataset is processed using the defined workflow. All preprocessing
 35 and quality control modules are executed sequentially on the full dataset,

1 ensuring that results reflect the finalized and verified configuration established
2 during preview.

3 Unlike interactive execution, analysis mode prioritizes correctness and
4 completeness over responsiveness. The pipeline is executed as a single,
5 continuous process, applying each transformation deterministically to the entire
6 dataset. This ensures that the final output is consistent with the validated pipeline
7 logic and suitable for downstream analysis or reporting.

8 In contrast to existing graphical tools such as SNPStats, which impose strict
9 runtime limitations on execution, such as 10 minutes, the Genomic Analyzer does
10 not enforce an artificial processing window. This allows the system to handle larger
11 datasets without interruption, enabling comprehensive analysis workflows that
12 would otherwise exceed the constraints of web-based environments.

13 14 Pipeline Ordering and Constraints

15 The pipeline supports flexible construction through user-defined ordering of
16 modules, while maintaining constraints that ensure valid execution semantics for
17 genomic data processing. This design allows users to iteratively build and refine
18 workflows without restricting interaction, while still preserving the logical
19 separation between transformation and evaluation stages. To achieve this, the
20 system defines ordering constraints that reflect domain requirements, alongside
21 mechanisms that allow users to dynamically modify pipeline structure during
22 workflow construction.

23 24 *Preprocessing-quality control sequencing*

25
26 The pipeline enforces a conceptual ordering in which preprocessing
27 operations are applied prior to quality control evaluation. Preprocessing modules
28 are responsible for transforming raw input data into a consistent internal
29 representation, while quality control modules evaluate and filter data based on
30 statistical and domain-specific criteria. Ensuring that quality control operates on
31 normalized data is essential for producing valid and interpretable metrics.

32 Although the underlying pipeline representation is structurally agnostic, this
33 sequencing is maintained to preserve correctness of execution. Applying quality
34 control only after preprocessing ensures that all evaluations are performed on
35 standardized and fully prepared datasets.

36 37 *User-driven reordering and insertion*

38
39 The system allows users to dynamically reorder and insert modules within
40 the pipeline during workflow construction. This capability supports iterative
41 refinement, enabling users to experiment with different transformation
42 sequences and configurations without rebuilding the pipeline from scratch.
43 Modules can be repositioned or added at arbitrary points, providing flexibility in
44 defining custom workflows tailored to specific datasets. While the backend
45 pipeline structure does not impose strict type-based constraints on module

1 placement, the user interface prevents modules from being in the incorrect
2 pipeline.

3 4 *Data Transformation Pipelines*

5
6 The Genomic Analyzer structures data processing as a sequence of
7 deterministic transformations applied through modular pipelines. The
8 preprocessing pipeline is responsible for resolving heterogeneity in input
9 datasets, converting diverse formats and encodings into a consistent internal
10 representation. This standardization step is essential for ensuring that
11 downstream quality control and analysis operations operate on well-defined and
12 comparable data.

13 14 Preprocessing Pipeline

15 The preprocessing pipeline transforms raw genomic datasets into a
16 normalized form by resolving inconsistencies in encoding, structure, and
17 representation. Because real-world genomic data often originates from multiple
18 sources with differing conventions, this stage is critical for ensuring correctness
19 and comparability across datasets. Each preprocessing operation is applied as a
20 discrete transformation, allowing the pipeline to incrementally enforce
21 consistency while preserving transparency of intermediate states.

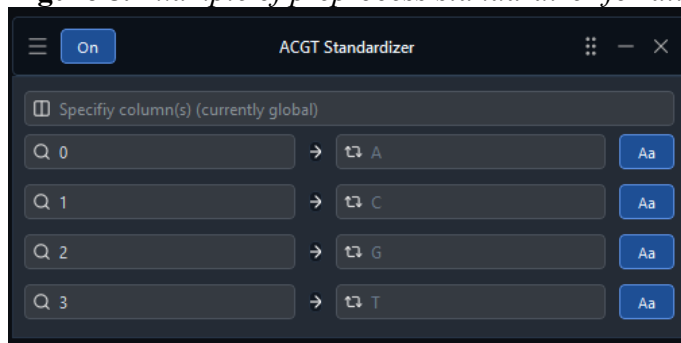
22 23 *Data normalization strategy*

24
25 The normalization strategy defines how heterogeneous input data is mapped
26 into a unified internal representation (see Figure 8). This process includes
27 standardizing genotype encodings, resolving missing value conventions, and
28 ensuring consistent structural alignment across rows and columns. Deterministic
29 transformation rules are enforced such that identical inputs and configurations
30 always produce identical normalized outputs.

31 Rather than assuming a fixed input schema, the normalization process is
32 designed to accommodate variation in dataset structure and encoding. This
33 allows the system to handle real-world variability while still converging on a
34 consistent representation required for downstream processing. By isolating
35 normalization as a dedicated stage, the pipeline ensures that all subsequent
36 operations operate on validated and standardized data.

37 Standardizers are built using a specialized variant of the find-and-replace
38 component. This variant accelerates the development as well as keeps the
39 standardization process consistent.

40

1 **Figure 8.** *Example of preprocess standardizer for alleles*

2
3 *Source:* Created by author.

4 *Genotype standardization*

5
6
7 Genotype data may be represented in multiple formats, including allele-
8 based encodings (e.g., A/A, A a), numeric encodings (e.g., 1/2, 12), or
9 concatenated representations (e.g., AA, Aa). These variations introduce
10 ambiguity in interpretation and must be resolved prior to analysis. The
11 preprocessing pipeline standardizes all genotype values into a unified
12 representation, ensuring that equivalent biological states are consistently
13 encoded across the dataset.

14 This standardization process is essential for enabling accurate computation
15 of downstream metrics such as minor allele frequency and Hardy–Weinberg
16 equilibrium. Without a consistent genotype representation, these calculations
17 would produce unreliable or inconsistent results. By enforcing a uniform
18 encoding, the system ensures that statistical operations are applied correctly and
19 comparably across all samples and variants.

20 For the genotype standardized module, the user is prompted for the 4 alleles
21 that they used in their dataset, and the module shows A, C, G and T as the
22 respective outputs, guiding input placement and expected value mapping (see
23 Figure 8). They also get to specify their delimiter used, or blank for none, and
24 the module either replaces, if the delimiter was different, or adds if a delimiter
25 was not used.

26 27 *Delimiter and format standardization*

28
29 Input datasets may use a variety of delimiters and formatting conventions,
30 including commas, tabs, spaces, or mixed separators. In addition, inconsistencies
31 such as irregular spacing or malformed rows can lead to parsing ambiguity. The
32 preprocessing pipeline normalizes these structural variations to enforce
33 consistent data alignment and column interpretation.

34 While this transformation is less complex than genotype normalization, it is
35 necessary for ensuring reliable data ingestion and preventing structural errors
36 during subsequent processing stages. By enforcing a uniform format early in the
37 pipeline, the system reduces the risk of downstream inconsistencies and
38 simplifies the execution of later transformations.

1 Quality Control Pipeline

2 The quality control pipeline evaluates the integrity of the normalized dataset
3 by computing statistical metrics and applying threshold-based filtering criteria.
4 These operations are designed to identify unreliable or uninformative data points
5 prior to downstream analysis. The separation of quality evaluation from
6 preprocessing ensures all metrics are computed on a consistent and validated
7 representation of the data.

8 Quality control is performed at multiple levels of granularity, including both
9 variant-level (SNP) and sample-level evaluation. Each metric captures a
10 different aspect of data quality, and together they provide a comprehensive
11 framework for filtering and validating genomic datasets.

12 *Minor allele frequency filtering*

13
14
15 Minor allele frequency (MAF) measures the proportion of the less common
16 allele for a given SNP across the dataset. Variants with extremely low frequency
17 may be uninformative for statistical analysis or may reflect sequencing errors
18 and insufficient representation within the sample population.

19 The system computes MAF for each SNP and applies a configurable
20 minimum threshold to exclude variants below the specified cutoff, commonly
21 used in GWAS workflows (Cordell and Clayton, 2005). This filtering step
22 reduces noise and improves the stability of downstream statistical models by
23 ensuring that retained variants have sufficient representation.

24 *Hardy-Weinberg equilibrium testing*

25
26
27 Hardy–Weinberg equilibrium (HWE) testing evaluates whether observed
28 genotype frequencies for a given SNP deviate from expected distributions under
29 equilibrium conditions. Significant deviations may indicate issues such as
30 genotyping errors, population stratification, or data inconsistencies.

31 The system performs statistical testing to compute an HWE p-value for each
32 SNP and applies a configurable threshold to exclude variants that significantly
33 deviate from equilibrium. This ensures that retained variants conform to
34 expected population-level behavior and reduces the risk of bias in downstream
35 analysis.

36 *Missingness filtering*

37
38
39 Missingness measures the proportion of absent or undefined genotype
40 values within the dataset and serves as a key indicator of data completeness.
41 High levels of missing data can reduce statistical power and introduce bias,
42 making it necessary to exclude incomplete observations.

43 The system supports both SNP-level and sample-level missingness
44 evaluation. SNP-level missingness captures the proportion of missing values for
45 each variant, while sample-level missingness captures the proportion of missing
46 values across all variants for a given sample. In both cases, thresholds are

1 configurable, allowing users to define acceptable levels of completeness based
2 on dataset characteristics and analysis requirements.

3 By supporting flexible, percentage-based thresholds at multiple levels of
4 granularity, the system provides precise control over data filtering while
5 maintaining consistency and transparency in quality control decisions. Since the
6 design allows for multiple modules of the same type, different percentages can
7 be assigned in a SNP-level module compared to a sample-level module.

8 9 Exclusion Logic and Decision Tracking

10 To maintain transparency, the system records detailed information for each
11 exclusion event, including the entity identifier, the metric values that triggered
12 the exclusion, the corresponding threshold, and the specific rule applied. This
13 structured tracking allows users to understand not only which data points were
14 removed, but also the exact reasoning behind each decision. By preserving this
15 information, the system enables users to audit the effects of quality control
16 operations and verify that filtering criteria have been applied correctly.

17 All exclusion decisions are incorporated into the system's reproducibility
18 framework, ensuring that the same inputs and configuration will produce
19 identical filtering outcomes. The recorded exclusion data can be included in
20 exported summaries and run manifests, providing a complete and traceable
21 record of how the dataset was transformed. This approach supports both
22 interpretability and reproducibility, allowing analysis workflows to be reviewed,
23 shared, and re-executed with confidence.

24 25 *Reproducibility and Configuration*

26
27 Reproducibility is critical in computational studies (Balagué-Dobón et al.,
28 2022). In existing workflows, preprocessing and quality control steps are often
29 configured manually and reconstructed from memory, notes, or terminal history,
30 making exact repetition difficult across time, users, and environments. The
31 system addresses this limitation by treating configuration as a first-class artifact
32 of execution, ensuring that the full state of a workflow can be preserved,
33 reloaded, and applied consistently.

34 Rather than limiting reproducibility to final outputs alone, the system
35 captures the structure and parameters of the workflow itself. This allows the
36 same pipeline to be reconstructed and executed deterministically, supporting
37 validation, collaboration, and long-term reuse. To achieve this, the system
38 combines persistent configuration storage with a run manifest that records how
39 a particular analysis was defined and executed.

40 41 Run Manifest Design

42 The run manifest serves as the primary reproducibility artifact of the system.
43 It records the effective configuration used for a given execution, including the
44 structure of the pipeline, the enabled modules, the parameter values associated
45 with each module, and the outputs generated by the run. By preserving this
46 information in a structured format, the system ensures that a completed analysis

1 can be understood and repeated without relying on external notes or manual
2 reconstruction.

3 This design allows the run manifest to function not only as an execution
4 record, but also as a portable description of the workflow itself. As a result,
5 analyses can be reviewed after completion, shared with collaborators, and re-
6 executed under the same conditions. The run manifest therefore supports both
7 immediate reproducibility and longer-term auditability of genomic processing
8 workflows.

9 10 *Configuration serialization*

11
12 Configuration serialization defines how workflow state is converted into a
13 persistent, machine-readable representation. Each module contributes its own
14 configurable state to the serialized output, allowing the full pipeline definition
15 to be preserved in a structured and modular form. This includes not only high-
16 level pipeline ordering, but also the detailed settings required to reproduce the
17 behavior of each transformation and quality control step.

18 By serializing configuration as structured data, the system avoids the
19 ambiguity and incompleteness of manual workflow documentation. This
20 approach also supports versioning and inspection, making configurations easier
21 to compare, store, and reuse across repeated analyses.

22 23 Module-level state persistence

24 Each module is responsible for persisting the state required to reproduce its
25 behavior. This may include threshold values, transformation parameters,
26 selected columns, enabled or disabled status, and any other module-specific
27 settings that influence execution. Because state persistence is handled at the
28 module level, reproducibility remains consistent even as new module types are
29 added to the system.

30 This design aligns with the modular architecture of the pipeline, ensuring
31 that each module remains self-contained not only in execution, but also in
32 configuration. As a result, the reproducibility framework scales naturally with
33 system extensibility.

34 35 Pipeline configuration export

36 In addition to preserving individual module state, the system exports the
37 structure of the pipeline itself, including module ordering and composition. This
38 allows an entire workflow to be represented as a single portable configuration
39 rather than a collection of disconnected parameter values. By capturing both
40 module configuration and pipeline structure, workflow semantics are preserved
41 during export.

42 This exported configuration can be stored independently of the original
43 execution environment, enabling reuse across sessions, datasets, and users. The
44 result is a reproducibility model in which workflows can be treated as persistent
45 analytical artifacts rather than temporary interface state.

1 Configuration rehydration

2 Configuration rehydration is the inverse of serialization, allowing
3 previously saved workflow definitions to be reconstructed within the system.
4 During rehydration, the pipeline structure is restored, module instances are
5 recreated, and persisted parameter values are reapplied to recover the original
6 analytical configuration.

7 This capability allows users to resume prior work, revisit earlier analyses,
8 or apply the same workflow to new datasets without rebuilding pipelines
9 manually. Rehydration therefore extends reproducibility beyond archival storage
10 by making saved configurations directly usable in future execution.

11 *Reproducible execution model*

12
13
14 The reproducible execution model ensures that identical inputs and identical
15 configurations produce identical outputs. Once a pipeline has been defined,
16 serialized, and executed, the same workflow can be reconstructed and reapplied
17 deterministically, preserving both transformation behavior and exclusion logic.
18 This guarantees that reproducibility is not limited to final reports, but is
19 embedded throughout the system's execution process.

20 By combining manifest-based execution records, module-level state
21 persistence, and deterministic pipeline behavior, the system establishes
22 reproducibility as a property of the architecture rather than an afterthought. This
23 is particularly important in genomic workflows, where preprocessing and
24 filtering decisions can significantly affect downstream results. Through explicit
25 configuration management, the Genomic Analyzer enables these decisions to
26 remain inspectable, repeatable, and transferable across time and context.

27 28 Output and Results Generation

29 The Genomic Analyzer produces structured outputs designed to support
30 both reproducibility and interpretability. Rather than treating outputs as isolated
31 artifacts, the system generates a coordinated set of machine-readable and human-
32 readable representations that capture both the results of analysis and the
33 configuration used to produce them. This ensures that workflows can be
34 reviewed, shared, and re-executed without loss of context.

35 Outputs are organized to preserve the relationship between configuration,
36 execution, and results. By coupling analysis outputs with their corresponding
37 configuration data, workflows are treated as portable and versionable artifacts
38 that can be reused across datasets and environments.

39 40 *Machine-readable configurations (JSON)*

41
42 Machine-readable outputs are generated in structured JSON format and
43 serve as the primary mechanism for reproducibility and configuration
44 persistence. These outputs include serialized pipeline configurations, module-
45 level parameters, and execution metadata required to reconstruct and re-execute
46 a workflow.

1 JSON configurations can be stored, shared, and version-controlled using
2 standard tools such as Git, allowing workflows to be tracked and compared over
3 time. This enables users to apply the same configuration to different datasets, or
4 to evaluate the impact of configuration changes on a fixed dataset. By
5 externalizing configuration in a portable format, the system supports repeatable
6 experimentation and collaborative analysis.

7 These configurations also enable direct rehydration within the system,
8 allowing previously defined workflows to be reconstructed and executed without
9 manual reconfiguration. As a result, JSON outputs function as both a
10 reproducibility mechanism and a reusable definition of analytical intent.

11 *Human-readable outputs (PDF, HTML)*

12
13
14 In addition to machine-readable outputs, the system generates human-
15 readable reports that summarize analysis results and configuration context.
16 These reports are produced in both PDF and HTML formats, providing
17 complementary modes of presentation while preserving the same underlying
18 information.

19 The PDF output serves as a complete, self-contained document that captures
20 the full results of an analysis in a single, portable format. This includes summary
21 statistics, filtering outcomes, and relevant configuration details, allowing the
22 document to be shared or archived independently of the system.

23 The HTML output provides an alternative representation that emphasizes
24 navigability and interactive exploration. Information is structured into
25 collapsible or paginated sections, allowing users to browse large result sets more
26 efficiently while maintaining access to the same content as the PDF. This dual-
27 format approach ensures that results are both easily consumable and fully
28 preserved, regardless of the medium through which they are accessed.

29 30 *Extensibility and System Evolution*

31
32 The Genomic Analyzer is designed to support incremental system evolution
33 through modular and extensible architecture. Rather than treating functionality
34 as fixed at implementation time, new capabilities can be introduced without
35 modifying existing components. This design ensures that the system can adapt
36 to new workflows, datasets, and analysis requirements while preserving stability
37 and reproducibility.

38 Extensibility is achieved through well-defined abstractions at both the
39 module and engine levels, allowing the system to grow in a controlled and
40 predictable manner. By isolating functionality within composable units and
41 standard interfaces, the architecture supports continuous expansion without
42 increasing system complexity disproportionately.

43 44 Module Extensibility Model

45 New functionality within the system is introduced through the addition of
46 modules, which encapsulate discrete transformation or evaluation operations.

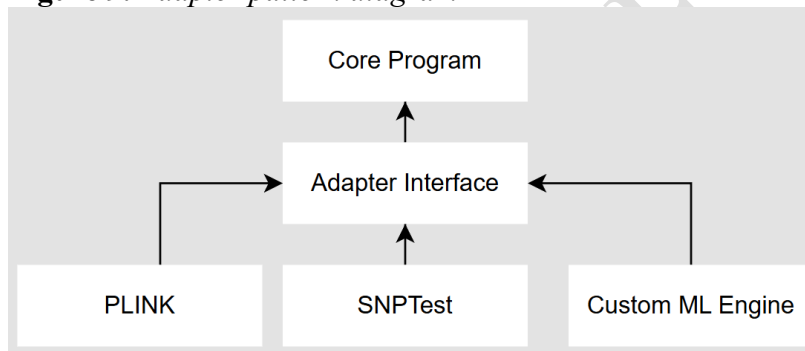
1 Because modules adhere to a shared abstraction, they can be integrated into
 2 existing pipelines without requiring changes to other components. This enables
 3 the system to expand organically as new preprocessing techniques, quality
 4 control metrics, or domain-specific operations are developed.

5 The compositional design of modules further accelerates extensibility by
 6 allowing new modules to be constructed from reusable internal components. As
 7 the library of these components grows, the effort required to implement
 8 additional modules decreases, enabling faster iteration and more consistent
 9 behavior across the system. This results in an extensibility model where system
 10 growth becomes progressively more efficient over time.

11 Engine Extensibility Model

12 The system supports extensibility at the analysis level through its adapter-
 13 based engine integration. New analysis engines can be incorporated by
 14 implementing the standard adapter interface, allowing them to interact with the
 15 pipeline without requiring modifications to the core system or existing
 16 workflows.
 17

18 **Figure 9.** *Adapter pattern diagram*



20 *Source:* Created by author.

21
 22
 23 This design enables the system to evolve alongside advancements in
 24 genomic analysis tools and methodologies. As new engines are introduced,
 25 including those based on alternative statistical approaches or machine learning
 26 techniques, they can be integrated seamlessly into the existing architecture. By
 27 maintaining a consistent interface, the system ensures that workflows remain
 28 stable even as underlying analysis capabilities expand. This results in analysis
 29 engines being decoupled from the core system, allowing them to be interchanged
 30 (see Figure 9).

31 Future Type System Integration

32 While the current system focuses on flexible and generalized data
 33 transformation, future iterations will introduce a more expressive internal type
 34 system to strengthen correctness and validation. This evolution will enable richer
 35 representations of genomic data, allowing transformations and quality control
 36 operations to operate on well-defined data types rather than generic
 37 representations.
 38

1 The introduction of a structured type system will support stronger
 2 guarantees around data integrity, reduce ambiguity in transformation logic, and
 3 improve the reliability of downstream computations. This enhancement is
 4 designed to integrate with the existing modular architecture, ensuring that type-
 5 aware behavior can be incorporated without disrupting established workflows or
 6 requiring significant changes to the pipeline structure.

7 8 9 Results

10 The Genomic Analyzer demonstrates several core capabilities aligned
 11 with its architectural design. Figure 10 illustrates the system interface, including
 12 the preprocessing pipeline, quality control modules, and real-time dataset
 13 preview.

14 **Figure 10.** *Preprocessing and quality control pipelines with real-time dataset*
 15 *preview*

The screenshot displays the Genomic Analyzer interface. On the left, the 'Preprocess Pipeline' section contains four modules: 'Find & Replace' (twice), 'Null Standardizer', and 'Delimiter Standardizer'. Each module has a 'Clear All' button and a 'Find & Replace' button. The 'Quality Control Pipeline' section on the right contains two modules: 'Row Missingness QC' and 'Duplicate ID QC'. The 'Row Missingness QC' module shows a 'Max missing ratio per row' of 0.95 and 'Rows: 277 - 277'. The 'Duplicate ID QC' module shows a message: 'Keeps the first row for each value in the selected ID column. Column not found.' On the far right, a real-time dataset preview table is shown with columns: ID, Affection, Sex, DBP1_1, DBP1_2, SENUM, SESSTAT, AnticCP, RFUW, rs10439884, rs2268810, and rs29491. The table contains 20 rows of data.

ID	Affection	Sex	DBP1_1	DBP1_2	SENUM	SESSTAT	AnticCP	RFUW	rs10439884	rs2268810	rs29491
D0034949	no	female	101	401	SS	yes	?	?	A,G	A,A	A,A
D0034332	?	Male	?	?	NN	yes	13	23	T,C	A,A	A,A
D0032151	no	?	101	?	SN	?	?	?	T,G	A,A	A,A
D0032042	?	?	102	2	NN	no	12	?	G,G	A,A	A,A
D0034949	no	female	?	?	SS	yes	?	?	A,G	A,A	A,A
D00321275	?	Female	?	?	NN	yes	?	34	A,T	A,A	A,A
D0032163	no	?	?	403	?	?	32	45	T,C	A,A	A,A
D0032795	?	?	?	3	NN	no	?	?	T,G	A,A	A,A
6945201	?	female	101	?	SN	yes	31	14	A,G	G,G	C,C
D0032027	no	Male	101	?	?	?	?	?	C,T	A,G	A,C
1915200	?	?	101	?	SN	yes	11	45	T,T	A,G	A,C
D0019941	yes	?	102	?	NN	?	?	?	A,G	A,G	A,C
D0016405	?	female	?	?	?	yes	?	?	T,T	A,G	A,C
D0034949	no	female	?	401	SS	?	?	?	A,G	A,A	A,A
D0034332	?	?	?	?	NN	yes	13	23	T,C	A,A	A,A
D0032151	no	?	?	?	SN	no	?	?	T,G	A,A	A,A
D0032042	?	male	102	?	NN	no	12	?	G,G	A,A	A,A
D00321275	yes	Female	101	?	NN	yes	?	34	A,T	A,A	A,A
D0032163	no	Male	?	?	SN	yes	32	45	T,C	A,A	A,A
D0032795	no	female	101	3	NN	no	?	?	T,G	A,A	A,A
6945201	yes	?	?	?	SN	?	31	14	A,G	G,G	C,C
D0032027	no	Male	101	?	SN	no	?	?	C,T	A,G	A,C

18
19 Source: Created by author.

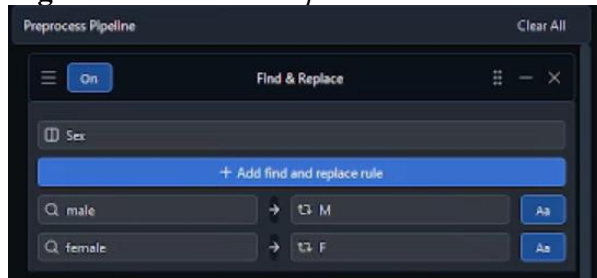
20
21 The current implementation of the Genomic Analyzer demonstrates the
 22 effectiveness of its modular architecture and interactive workflow design. Initial
 23 development of the first preprocessing module required approximately ten hours,
 24 primarily to establish the graphical interface, component structure, and
 25 supporting framework. In contrast, subsequent modules were implemented
 26 significantly faster. The quality control pipeline, including multiple modules,
 27 was developed in under one hour, illustrating a substantial reduction in
 28 development effort as reusable components and architectural patterns were
 29 leveraged.

30 This improvement is largely driven by component reuse within the
 31 preprocessing system. For example, the allele standardization module was
 32 constructed by reusing an existing substitution component with constrained
 33 parameters, as well as utilizing the module factory, which builds the title bar,
 34 background and graphical interface. Thus, the construction of the allele module
 35 required minimal additional implementation effort. This component is used

1 across multiple modules, including the find-and-replace module and future
2 standardization tasks.

3

4 **Figure 11.** *Find-and-replace module*



5

6 *Source:* Created by author.

7

8 The find-and-replace module utilizes an indefinite instance count of the
9 substitute component, which is appendable by the user at runtime (see Figure
10 11). This illustrates another benefit of modularity, including dynamic addition
11 and removal of modules and components.

12 The effort required to implement new modules decreases as the system
13 evolves, demonstrating a non-linear development pattern where system
14 expansion becomes progressively more efficient.

15 The system also provides real-time dataset transformation through an
16 interactive preview mechanism. Users can observe transformations as they are
17 applied, with changes reflected immediately within the dataset view (see Figure
18 10). This allows for continuous validation of preprocessing logic, including
19 detection of unintended transformations such as ambiguous token replacements.
20 Unlike traditional workflows that require full execution before results can be
21 inspected, the preview system enables users to verify correctness incrementally,
22 reducing the likelihood of propagating configuration errors.

23

24

25 **Discussion**

26

27 The observed development patterns highlight the effectiveness of the
28 system's modular architecture. By decomposing functionality into reusable
29 components, rapid construction of new modules without requiring modification
30 of existing code is enabled. This aligns with established software design
31 principles, including the open-closed principle, where new functionality is
32 introduced through extension rather than modification. As the number of
33 available components increases, the effort required to implement additional
34 modules decreases, resulting in a development process that becomes more
35 efficient over time.

36

37 The preview execution model represents a significant departure from
38 traditional genomic analysis workflows. Existing tools such as SNPStats and
39 PLINK rely on batch-oriented execution, where preprocessing configurations
40 are defined prior to execution and results are only available after completion. In
contrast, the Genomic Analyzer allows users to observe transformations in real

1 time, providing immediate feedback and enabling iterative refinement of
2 preprocessing steps. This reduces both cognitive overhead and debugging time,
3 while improving overall workflow transparency.

4 This preview execution model was not initially designed as a user-facing
5 feature, but emerged during development as an internal debugging tool. While
6 implementing preprocessing transformations, it became necessary to visualize
7 intermediate dataset states to verify correctness. The immediate utility of this
8 capability revealed a broader usability gap in existing tools, where users must
9 often execute full workflows before observing results. By exposing this
10 functionality within the interface, the system transforms a developer-centric
11 debugging mechanism into a core user-facing feature, enabling real-time
12 validation and significantly improving the overall workflow.

13 The system's modular isolation further contributes to its extensibility.
14 Modules operate independently and are agnostic to their position within the
15 pipeline, requiring no knowledge of upstream or downstream components. This
16 design allows new modules to be introduced without impacting existing
17 functionality, supporting scalable system growth. However, the current
18 implementation is limited to preprocessing functionality, with quality control
19 functionality partially implemented and continuing to be expanded. While the
20 architecture is designed to support these extensions, further work is required to
21 fully realize this capability and evaluate system performance on large-scale
22 datasets.

23 24 25 **Conclusion**

26
27 This paper presents the Genomic Analyzer, a modular framework for
28 genomic data preprocessing designed to improve usability, reproducibility, and
29 extensibility. The system demonstrates that a component-based architecture
30 enables rapid development of new functionality, with implementation effort
31 decreasing as reusable components accumulate. The integration of real-time
32 dataset preview further enhances workflow transparency, allowing users to
33 validate transformations incrementally and reduce configuration errors.

34 By combining modular design with interactive execution, the system
35 addresses key limitations in existing tools, including rigid workflows, lack of
36 intermediate validation, and reliance on external command-line processes. While
37 the current implementation focuses on preprocessing, the architecture provides
38 a foundation for integrating quality control modules and analysis engines in
39 future work. Continued development will focus on expanding system
40 capabilities and evaluating performance across larger and more complex
41 genomic datasets.

42

References

- 1
2
3 Shaun Purcell (2009) PLINK Tutorial, University of Colorado Boulder,
4 [https://ibgwww.colorado.edu/cdrom2009/ScriptsA/purcell/GWAS/boulder_09_plink_i](https://ibgwww.colorado.edu/cdrom2009/ScriptsA/purcell/GWAS/boulder_09_plink_intro.pdf)
5 [ntro.pdf](https://ibgwww.colorado.edu/cdrom2009/ScriptsA/purcell/GWAS/boulder_09_plink_intro.pdf)
6 Jonathan Marchini, Bryan Howie (2010) SNPTEST: software for genome-wide
7 association studies, University of Oxford,
8 https://mathgen.stats.ox.ac.uk/genetics_software/snptest/snptest_v2.4.1.html
9 SNPStats (2026) SNPStats preprocessing interface, <https://www.snpstats.net/preproc.php>
10 Azza M. Al-Khalifa, Mohamed N. Saad, Muhammad A. Rushdi, and Ayman M. Eldeib
11 (2023) Multithreaded Haplotype Block Partitioning of Rheumatoid Arthritis
12 Genomic Data, 2023 IEEE EMBS Special Topic Conference on Data Science and
13 Engineering in Healthcare, Medicine and Biology, December 7-9, 2023. Malta,
14 <https://ieeexplore.ieee.org/document/10404920>
15 Gang Zheng, Yaning Yang, Xiaofeng Zhu, Robert C. Elston (2012) Analysis of Genetic
16 Association Studies, Springer New York, NY, ISBN 978-1-4614-2244-0, 1st
17 Edition, 2012, <https://doi.org/10.1007/978-1-4614-2245>
18 Cordell HJ, Clayton DG (2005) Genetic association studies, The Lancet, September 24-
19 30, 2005, 366 (9491), 1121-1131, [https://doi.org/10.1016/S0140-](https://doi.org/10.1016/S0140-6736(05)67424-7)
20 [6736\(05\)67424-7](https://doi.org/10.1016/S0140-6736(05)67424-7), <https://pubmed.ncbi.nlm.nih.gov/16182901/>
21 Laura Balagué-Dobón, Alejandro Cáceres and Juan R. González (2022) Fully exploiting
22 SNP arrays: a systematic review on the tools to extract underlying genomic
23 structure, Briefings in Bioinformatics, 23(2), 1-22, [https://doi.org/10.1093/](https://doi.org/10.1093/bib/bbac043)
24 [bib/bbac043](https://doi.org/10.1093/bib/bbac043)
25 Christopher C Chang, Carson C Chow, Laurent CAM Tellier, Shashaank Vattikuti,
26 Shaun M Purcell, James J Lee (2015) Second-generation PLINK: rising to the
27 challenge of larger and richer datasets, GigaScience, 4(1), [https://doi.org/10.1186/](https://doi.org/10.1186/s13742-015-0047-8)
28 [s13742-015-0047-8](https://doi.org/10.1186/s13742-015-0047-8)
29 Chen, B., Wilkening, S., Drechsel, M. et al (2009) SNP_tools: A compact tool package
30 for analysis and conversion of genotype data for MS-Excel, BMC Research Notes
31 2, Article 214, <https://doi.org/10.1186/1756-0500-2-214>
32 Xavier Solé, Elisabet Guinó, Joan Valls, Raquel Iniesta, Víctor Moreno (2006)
33 SNPStats: a web tool for the analysis of association studies, Bioinformatics,
34 22(15), 15, 1928–1929, <https://doi.org/10.1093/bioinformatics/btl268>
35