Genomic Analyzer: A Scalable and High-Performance Object-Oriented Application for SNP Analysis and Genetic Association Studies

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

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

Introduction

SNP datasets are widely used in GWAS, yet they often originate from heterogeneous sources with inconsistent formats. This variability introduces significant preprocessing overhead and increases the likelihood of errors. Existing tools, such as PLINK (Purcell et al. 2009; Chang et al. 2015) and SNPTest (Marchini et al. 2010), require Command-line Interaction (CLI) which presents a steep learning curve and limits accessibility for new researchers.

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

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

objective of this work is to develop a modular, extensible genomic analysis system that eliminates manual preprocessing steps, improves reproducibility, and provides an intuitive graphical interface for users.

Literature Review

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

Graphical tools such as SNPStats (Solé et al. 2006; SNPStats 2026) attempt to improve usability but introduce other limitations, including restricted execution time and limited extensibility. These systems do not support modular pipeline construction, making it difficult to adapt workflows to new requirements.

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

Genome-Wide Association Studies and Tooling Requirements

Genome-wide association studies (GWAS) are a fundamental method in computational biology used to identify associations between genetic variants and observable traits or diseases. These studies rely on large datasets containing single nucleotide polymorphisms (SNPs), which represent small variations in DNA sequences across individuals. By analyzing these variations across populations, researchers can detect statistically significant correlations between specific genetic markers and phenotypic outcomes (Al-Khalifa et al. 2023; Zheng et al. 2012; Cordell et al. 2005).

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

As a result, GWAS tools must support complex data transformation pipelines, statistical analysis, and reproducibility tracking. However, these requirements introduce significant usability and workflow challenges. Tools must balance computational rigor with accessibility, while also providing

mechanisms for validating intermediate transformations and ensuring that analysis steps can be reproduced across different environments and timeframes. The design of effective GWAS tooling therefore lies at the intersection of domain-specific requirements and general software engineering principles.

Minor-Alle Frequency

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

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

Hardy-Weinberg Equilibrium

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

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

Missingness Rates

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

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

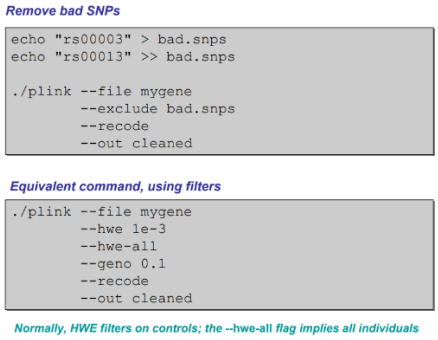
PLINK CLI Tool

PLINK is a widely used command-line toolset for genome-wide association studies (GWAS) and population-based genetic analysis (Chang et al., 2015). It provides functionality for data management, statistical association testing, and computation of quality control (QC) metrics such as minor allele frequency (MAF), Hardy–Weinberg equilibrium (HWE), and missingness rates.

How PLINK Works

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

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



Source: Purcell 2009.

PLINK Limitations

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

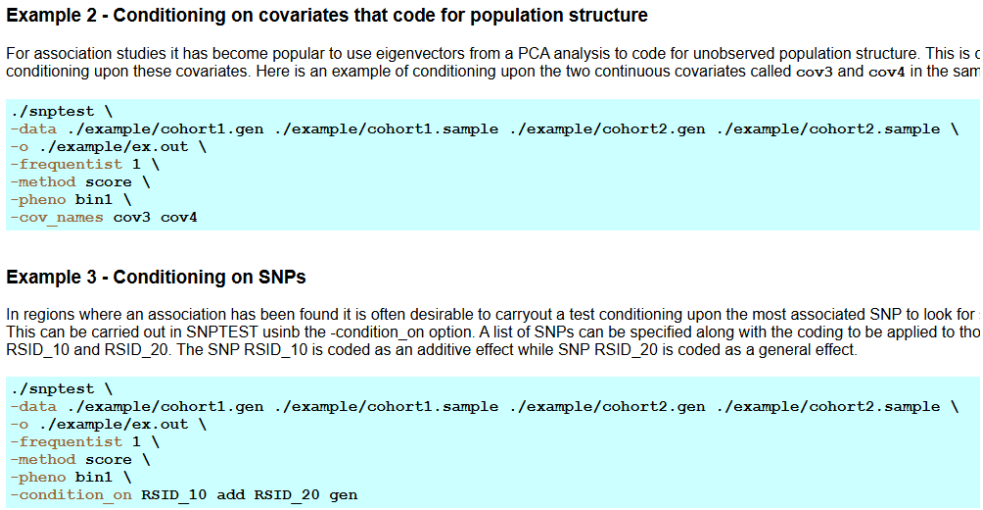
SNPTest CLI Tool

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

How SNPTest Works

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

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 c conditioning upon these covariates. Here is an example of conditioning upon the two continuous covariates called cov3 and cov4 in the sam

```
./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 usinb the -condition_on option. A list of SNPs can be specified along with the coding to be applied to tho 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
```

Source: Marchini et al. 2010.

SNPTest Limitations

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

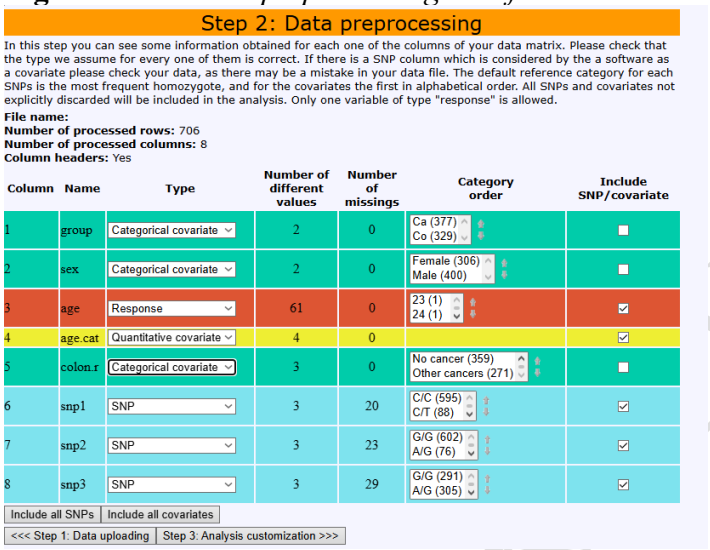
SNPStats GUI Tool

SNPStats is a web-based graphical tool designed to support genome-wide association studies by providing statistical analysis and visualization of SNP data (Solé et al. 2006; SNPStats 2026). It offers functionality for computing association metrics, performing quality control analyses, and generating human-readable outputs such as tables and graphical summaries in PDF format. Compared to command-line tools, SNPStats aims to improve accessibility through a graphical user-interface (GUI).

How SNPStats Works

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

Figure 3. *SNPStats preprocessing interface.*



Source: SNPStats 2026.

SNPStats Limitations

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

Preprocess steps and process

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

Platform constraints

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

Outdated application design

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

SNP Stats Overview

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

Comparative Analysis of Existing Tools

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

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

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

Methodology

System Architecture

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

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

Modular System Design

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

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

Module Abstraction

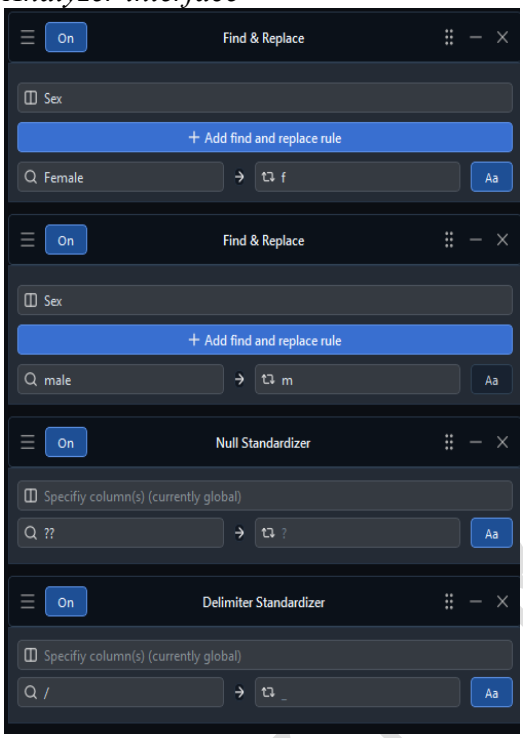
Modules are defined as the fundamental computational units within the system, each representing a discrete operation applied to genomic data. The module abstraction enforces a consistent interface for execution, allowing heterogeneous operations to be composed within a unified workflow. This abstraction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

By isolating functionality within modules, this design allows individual operations to be developed, tested and extended independently. This separation of concerns improves code maintainability and supports clear reasoning about data transformations, as each module is responsible for a well-defined limited scope behavior. This design allows for expandable info-cards at the title-bar or

side-menu of each module. This feature allows the interface itself to serve as an educational aid.

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

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



Source: Created by authors.

Preprocessing modules

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

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

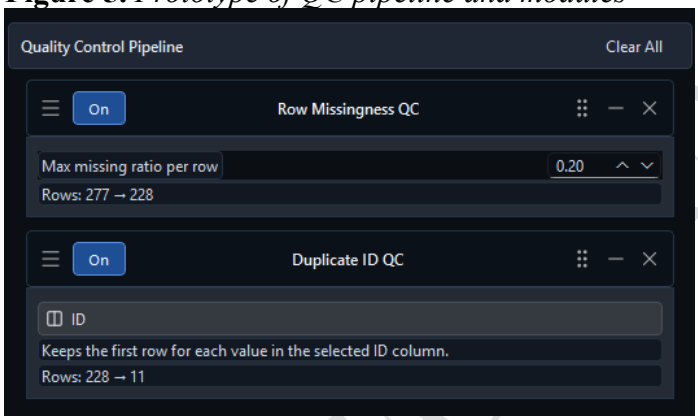
The default scope of preprocess modules is a global scope, meaning the module's changes will affect all columns of the dataset. Preprocess modules have

the option to be scoped to any column(s) (see Figure 4). This further allows greater flexibility, as some transformations would be invalid or unwanted in some columns.

Quality control modules

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

Figure 5. *Prototype of QC pipeline and modules*



Source: Created by authors.

Module composition and reusable components

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

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

By promoting reuse at the component level, the system reduces the complexity of developing new modules over time. As the library of reusable

components expands, the effort required to implement additional functionality decreases, allowing new modules to be constructed more rapidly and with greater consistency. This results in an extensibility model in which system growth becomes progressively more efficient, rather than increasingly complex.

Module discoverability and interaction model

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

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

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

Adapter-Based Engine Integration

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

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

By abstracting engine-specific behavior behind adapters, the system achieves engine agnosticism, enabling workflows to be constructed

independently of the chosen analysis backend. This allows users to configure and validate preprocessing and quality control pipelines without committing themselves to a specific execution environment. Engine selection becomes a runtime decision, where the same pipeline can be executed against different backends depending on the desired analysis method or output format.

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

Pipeline Architecture

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

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

Pipeline Execution Model

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

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

Preview mode (interactive execution)

In existing genomic analysis workflows, preprocessing and quality control pipelines are often constructed without immediate validation, increasing the likelihood of undetected errors or malformed data. If such issues are not identified early, they may propagate through the pipeline and result in incorrect downstream analysis. Because full dataset execution can be computationally

expensive, these errors are often discovered only after significant time has been spent processing invalid configurations, leading to inefficient and potentially misleading results.

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

Preview

ID	Affection	Sex	DRB1_1	DRB1_2	SENum	SESstatus	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

Source: Created by authors.

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

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

Subset based execution

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

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

Incremental re-computation strategy

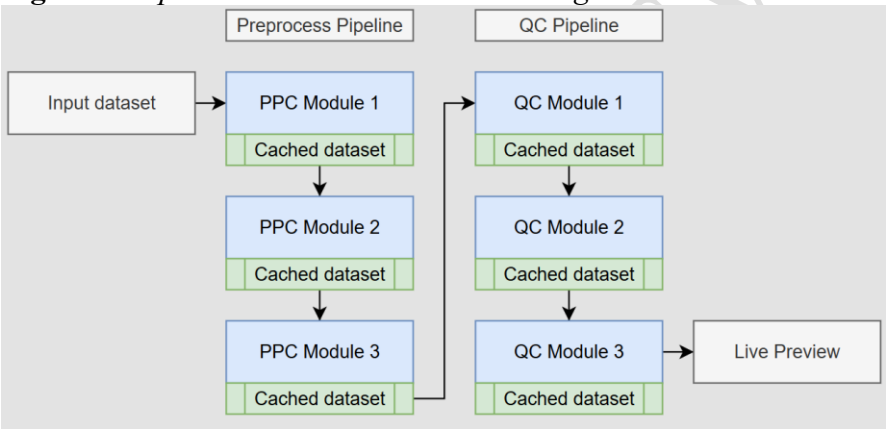
To further optimize interactive performance, the system employs an incremental re-computation strategy in which only affected portions of the pipeline are re-executed when changes are made. When a module is modified, all preceding modules remain unchanged, and only the modified module and its successors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

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

Intermediate state inspection

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

Figure 7. *Pipeline structure and data caching schema*



Source: Created by authors.

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

Analysis mode (final execution)

Once a pipeline has been fully constructed and validated by the user, the user initiates analysis using the configured pipeline. After the analysis is set up, the complete dataset is processed using the defined workflow. All preprocessing and quality control modules are executed sequentially on the full dataset, ensuring that results reflect the finalized and verified configuration established during preview.

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

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

Pipeline Ordering and Constraints

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

.

Preprocessing-quality control sequencing

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

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

User-driven reordering and insertion

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

Data Transformation Pipelines

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

Preprocessing Pipeline

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

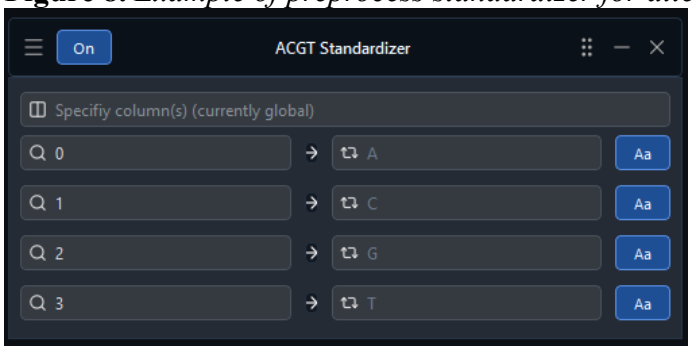
Data normalization strategy

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

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

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

Figure 8. *Example of preprocess standardizer for alleles*



Source: Created by authors.

Genotype standardization

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

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

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

Delimiter and format standardization

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

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

Quality Control Pipeline

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

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

Minor allele frequency filtering

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

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

Hardy-Weinberg equilibrium testing

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

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

Missingness filtering

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

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

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

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

Exclusion Logic and Decision Tracking

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

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

Reproducibility and Configuration

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

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

Run Manifest Design

The run manifest serves as the primary reproducibility artifact of the system. It records the effective configuration used for a given execution, including the structure of the pipeline, the enabled modules, the parameter values associated with each module, and the outputs generated by the run. By preserving this

information in a structured format, the system ensures that a completed analysis can be understood and repeated without relying on external notes or manual reconstruction.

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

Configuration serialization

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

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

Module-level state persistence

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

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

Pipeline configuration export

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

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

Configuration rehydration

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

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

Reproducible execution model

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

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

Output and Results Generation

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

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

Machine-readable configurations (JSON)

Machine-readable outputs are generated in structured JSON format and serve as the primary mechanism for reproducibility and configuration persistence. These outputs include serialized pipeline configurations, module-

level parameters, and execution metadata required to reconstruct and re-execute a workflow.

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

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

Human-readable outputs (PDF, HTML)

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

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

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

Extensibility and System Evolution

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

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

Module Extensibility Model

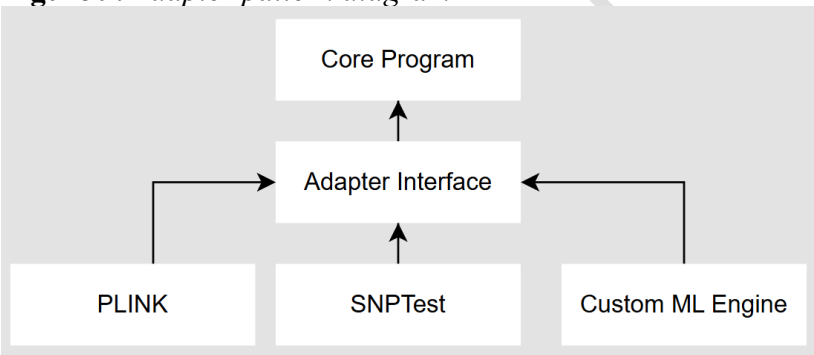
New functionality within the system is introduced through the addition of modules, which encapsulate discrete transformation or evaluation operations. Because modules adhere to a shared abstraction, they can be integrated into existing pipelines without requiring changes to other components. This enables the system to expand organically as new preprocessing techniques, quality control metrics, or domain-specific operations are developed.

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

Engine Extensibility Model

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

Figure 9. *Adapter pattern diagram*



Source: Created by authors.

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

Future Type System Integration

While the current system focuses on flexible and generalized data transformation, future iterations will introduce a more expressive internal type of system to strengthen correctness and validation. This evolution will enable

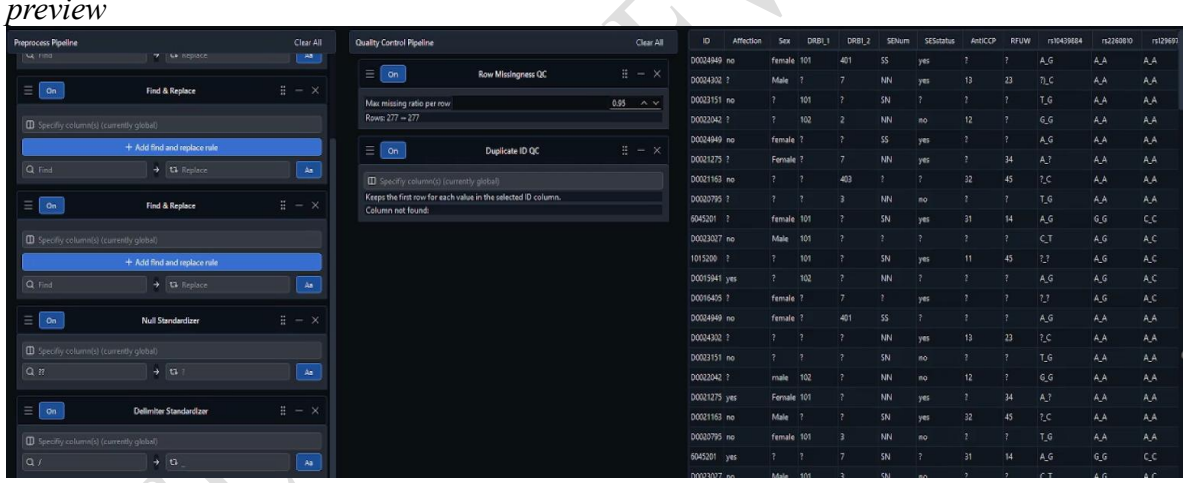
richer representations of genomic data, allowing transformations and quality control operations to operate on well-defined data types rather than generic representations.

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

Results

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

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



Source: Created by author.

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

This improvement is largely driven by component reuse within the preprocessing system. For example, the allele standardization module was constructed by reusing an existing substitution component with constrained parameters, as well as utilizing the module factory, which builds the title bar,

background and graphical interface. Thus, the construction of the allele module required minimal additional implementation effort. This component is used across multiple modules, including the find-and-replace module and future standardization tasks.

Figure 11. *Find-and-replace module*

Source: Created by authors.

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

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

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

Discussion

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

The preview execution model represents a significant departure from traditional genomic analysis workflows. Existing tools such as SNPStats and PLINK rely on batch-oriented execution, where preprocessing configurations

are defined prior to execution and results are only available after completion. In contrast, the Genomic Analyzer allows users to observe transformations in real time, providing immediate feedback and enabling iterative refinement of preprocessing steps. This reduces both cognitive overhead and debugging time, while improving overall workflow transparency.

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

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

Conclusion

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

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

References

Shaun Purcell (2009) PLINK Tutorial, University of Colorado Boulder, https://ibgwww.colorado.edu/cdrom2009/ScriptsA/purcell/GWAS/boulder_09_plink_intro.pdf

Christopher C Chang, Carson C Chow, Laurent CAM Tellier, Shashaank Vattikuti, Shaun M Purcell, James J Lee (2015) Second-generation PLINK: rising to the challenge of larger and richer datasets, GigaScience, 4(1), https://doi.org/10.1186 /s13742-015-0047-8

Jonathan Marchini, Bryan Howie (2010) SNPTEST: software for genome-wide association studies, University of Oxford, https://mathgen.stats.ox.ac.uk/genetics_software/snptest/snptest_v2.4.1.html

Azza M. Al-Khalifa, Mohamed N. Saad, Muhammad A. Rushdi, and Ayman M. Eldeib (2023) Multithreaded Haplotype Block Partitioning of Rheumatoid Arthritis Genomic Data, 2023 IEEE EMBS Special Topic Conference on Data Science and Engineering in Healthcare, Medicine and Biology, December 7-9, 2023. Malta, https://ieeexplore.ieee.org/document/10404920

Gang Zheng, Yaning Yang, Xiaofeng Zhu, Robert C. Elston (2012) Analysis of Genetic Association Studies, Springer New York, NY, ISBN 978-1-4614-2244-0, 1st Edition, 2012, https://doi.org/10.1007/978-1-4614-2245

Cordell HJ, Clayton DG (2005) Genetic association studies, The Lancet, September 24-30, 2005, 366 (9491), 1121-1131, https://doi.org/10.1016/S0140-6736(05)67424-7, https://pubmed.ncbi.nlm.nih.gov/16182901/

Chen, B., Wilkening, S., Drechsel, M. et al (2009) SNP_tools: A compact tool package for analysis and conversion of genotype data for MS-Excel, BMC Research Notes 2, Article 214, https://doi.org/10.1186/1756-0500-2-214

Laura Balagué-Dobón, Alejandro Cáceres and Juan R. González (2022) Fully exploiting SNP arrays: a systematic review on the tools to extract underlying genomic structure, Briefings in Bioinformatics, 23(2), 1-22, https://doi.org/10.1093/bib/bbac043

Xavier Solé, Elisabet Guinó, Joan Valls, Raquel Iniesta, Víctor Moreno (2006) SNPStats: a web tool for the analysis of association studies, Bioinformatics, 22(15), 15, 1928–1929, https://doi.org/10.1093/bioinformatics/btl268

SNPStats (2026) SNPStats preprocessing interface, https://www.snpstats.net/preproc.php