National College of Ireland

Performance Optimization and Cost Analysis of qpAdm Genomic Modeling on AWS Cloud Infrastructure

MSc Research Project
Cloud Computing

Utkarsh Singh
Student ID: 23332140

School of Computing
National College of Ireland

Supervisor: Sai Guna Ranjan Emani

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Utkarsh Singh

Student ID: 23332140

Programme: MSc Cloud Computing **Year:** …2025……..

Module: Research Project

Supervisor: Sai Guna Ranjan Emani

Submission Due Date: 2nd January 2026

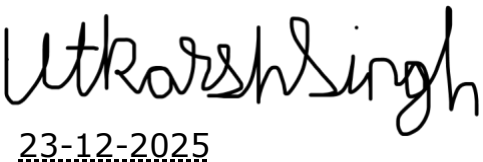
Project Title: Performance Optimization and Cost Analysis of gpAdm Genomic Modeling on AWS Cloud Infrastructure

Word Count: 10890

Page Count: 25

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: 

Date: 23-12-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Performance Optimization and Cost Analysis of qpAdm Genomic Modeling on AWS Cloud Infrastructure

Utkarsh Singh
23332140

Abstract

In the rapidly evolving field of paleogenomics, the analysis of ancient DNA (aDNA) has become a cornerstone of understanding human population history. Genomic analyses increasingly rely on computationally intensive workflows that integrate large genotype matrices with sophisticated statistical methods. Among these, qpAdm and related f-statistics tools from the ADMIXTOOLS suite are central to modelling population admixture, yet there is limited guidance on how to deploy them efficiently and reproducibly in cloud environments. This thesis investigates how to optimise the cost–performance trade-offs of qpAdm workflows on Amazon Web Services (AWS), while preserving reproducibility and scalability. A fully containerised pipeline was developed that encapsulates ADMIXTOOLS, PLINK and associated utilities within a version-pinned Docker image, deployed on EC2 instances provisioned via Terraform and driven by a lightweight benchmark runner inspired by Snakemake. Three representative datasets were used: a subset of the Allen Ancient DNA Resource (AADR) 1240k panel, a South Asian subset of the 1000 Genomes Project on chromosome 22, and a smaller synthetic dataset derived from SGDP-like structure. Experiments evaluated Graviton-based c7g, m7g and r7g instance families under on-demand and Spot pricing, with multiple repetitions per configuration and statistical analysis of log-transformed runtimes. The results show that preprocessing and format conversion dominate end-to-end runtime for non-EIGENSTRAT data, while qpAdm and qpfstats computations are primarily CPU-bound and scale well on compute-optimised Graviton3 hardware. c7g instances consistently provided the best price–performance among the evaluated families, and Spot instances reduced estimated compute costs by around two-thirds without compromising analytical validity. For the full AADR v62 1240k panel, we observed qpfstats consuming more than 31 GiB of memory and triggering out-of-memory termination on r7g.xlarge, motivating the use of r7g.2xlarge for the largest workloads. The pipeline, together with the empirical findings and derived heuristics, provides a practical blueprint for cost-effective, reproducible qpAdm modelling on AWS.

Keywords: ancient DNA; population admixture; qpAdm; f-statistics; ADMIXTOOLS; Amazon Web Services (AWS); cloud benchmarking; Graviton3; EC2; Spot Instances; Docker; Terraform; reproducible workflows; preprocessing; EIGENSTRAT; PLINK; cost–performance optimisation.

1 Introduction

1.1. Research Background

Genomic Ancient DNA has transformed the study of human population history by enabling direct observation of genetic variation in past populations. Advances in sequencing and enrichment technologies have produced large curated resources such as the Allen Ancient DNA Resource (AADR), which aggregates thousands of ancient and present-day individuals genotyped at dense SNP panels (Mallick et al., 2024). At the same time, modern reference panels such as the 1000 Genomes Project provide genome-wide variation in diverse populations, enabling joint analysis of ancient and contemporary data.

Within this landscape, qpAdm and related f-statistic methods from ADMIXTOOLS have become standard tools for modelling admixture histories (Patterson et al., 2012; Harney et al., 2021). These methods work by computing matrices of f-statistics that capture allele frequency correlations among populations and by fitting admixture models that explain a target population as a mixture of candidate sources relative to an outgroup set. The methodological developments in ADMIXTOOLS 2, including more efficient implementations and caching strategies (Maier et al., 2023), reduce redundant computation but do not in themselves solve the practical challenges of scaling workflows across large datasets and complex model grids.

As dataset sizes increase and studies demand extensive model exploration, the computational and logistical burden of running qpAdm pipelines becomes a major constraint. Researchers must convert heterogeneous genotype formats to EIGENSTRAT, harmonise population labels, repeatedly subset large matrices and manage numerous intermediate files. These tasks are often I/O-bound and poorly documented in the literature. At the same time, cloud platforms such as AWS offer elastic compute, diverse storage options and pricing models that could, in principle, support scalable and cost-efficient genomic analysis. Realising this potential, however, requires careful workflow design and infrastructure engineering.

1.2. Problem Statement

While qpAdm and f-statistic methods are methodologically mature and widely used, there is limited empirical guidance on how to execute them efficiently and reproducibly in a cloud environment. Several concrete problems arise in practice. First, cost–performance trade-offs for different EC2 instance families are unclear. It is not obvious a priori whether compute-optimised (c7g), general-purpose (m7g) or memory-optimised (r7g) instances provide the best balance between runtime and cost for typical qpAdm workloads. Mis-sizing instances may waste budget without materially improving turnaround time.

Second, preprocessing bottlenecks are often more severe than the qpAdm runs themselves. Converting large VCF or PLINK datasets to EIGENSTRAT, performing genotype transpositions and merging datasets with complex SNP intersections can dominate runtime and stress I/O subsystems. Third, traditional ADMIXTOOLS workflows recompute f-statistics from genotype files for each set of qpAdm models, leading to heavy I/O and redundant computation. Fourth, many existing pipelines rely on ad hoc shell scripts, manually configured cloud instances and weakly documented environments, which undermines reproducibility and hinders sharing of best practices.

There is therefore a need for a systematic, cloud-native pipeline that treats cost, performance and reproducibility as primary design objectives, rather than by-products of statistical analysis.

1.3. Research Question

To tackle the above problem, we pose the following overarching research question:

How can we optimize the cost–performance trade-offs of genomic modeling workflows (including preprocessing and analysis with qpAdm and qpFstats) on AWS cloud infrastructure, while ensuring reproducibility and scalability?

This question is unpacked into five sub-questions. First, it explores which components—preprocessing, qpfstats cache generation, or qpAdm execution—dominate runtime across different dataset scales. Second, it evaluates the cost-efficiency of various EC2 instance families, investigating how resource characteristics interact with input size and model complexity. Third, the impact of data representation is assessed, comparing standard and

packed EIGENSTRAT formats on runtime, memory consumption, and downstream cost. Fourth, the research investigates how technologies such as Docker, Snakemake, and Terraform can be orchestrated to yield a portable, maintainable pipeline. Finally, it seeks to translate empirical findings into practical guidelines that can inform genomic analysis at scale in the cloud.

1.4. Research Objective

The primary aim of this research is to deliver a validated, reproducible pipeline for cloud-based qpAdm benchmarking, supported by empirical data on runtime, resource usage, and cost. This begins with the creation of a Docker-based environment encapsulating ADMIXTOOLS, PLINK, and preprocessing logic. A Snakemake workflow is then developed to coordinate all stages—format conversion, population labeling, cache generation, and admixture modeling—within containers.

On the infrastructure side, a Terraform-managed setup provisions Graviton-based EC2 instances, deploys file systems such as FSx Lustre, and automates data synchronization with Amazon S3. This platform allows for controlled benchmarking across multiple instance families and dataset sizes, including the SGDP subset, the 1000 Genomes chromosome 22 subset, and the full AADR 1240k panel. Each run captures detailed metrics on runtime, memory usage, cache reuse, and per-run cost.

The analysis phase consolidates these metrics to assess performance and draw practical recommendations. The outcome is both a reproducible artifact—a cloud-native, end-to-end genomic pipeline—and a set of actionable insights that address current pain points in cloud genomics.

1.5. Research Contributions

This MSc research will yield several contributions to the intersection of bioinformatics, cloud computing, and qpAdm-based genomic modelling. It provides a fully containerised, infrastructure-as-code-driven pipeline for qpAdm workflows that can be redeployed across AWS accounts and local environments with minimal configuration changes. It offers a quantitative characterisation of runtime contributions from preprocessing, f-statistic caching and qpAdm runs, highlighting the central role of data format and I/O. It presents empirical evidence on the price–performance of Graviton3 instances for population genetics workloads, demonstrating that c7g instances typically deliver the most favourable cost per qpAdm experiment in the scenarios considered. It introduces a simple but extensible analysis toolkit for aggregating metrics, enriching them with scenario-based cost assumptions and applying basic statistical models. Finally, it distils these findings into a set of actionable heuristics for practitioners designing cloud-native qpAdm pipelines.

2. Related Work

This chapter situates the thesis within four overlapping strands of literature: statistical tools for admixture and population history, genotype data formats and infrastructure for large panels, cloud-based genomics pipelines and cost optimisation, and reproducible workflow and infrastructure-as-code practices. Together, these strands provide the scientific and engineering context for treating

qpAdm-based workflows as cloud workloads whose performance and cost can be rigorously evaluated.

2.1. Population Genetics Tools for Admixture Modeling

Statistical tools based on f-statistics have become central to the study of human population history, particularly in ancient DNA research. The ADMIXTOOLS suite introduced by Patterson et al. formalised the use of $f_3$and $f_4$statistics to detect admixture, quantify ancestry proportions, and test competing demographic models in a coherent framework (Patterson et al., 2012). Within this ecosystem, qpAdm has emerged as a workhorse method for modelling a target population as a mixture of candidate sources, constrained by a panel of outgroup populations. Its popularity stems from its interpretability and from its ability to operate on curated SNP panels, such as the Human Origins and 1240k datasets, rather than requiring whole-genome sequence data.

Subsequent work has examined the statistical properties and practical limitations of qpAdm in more detail. Harney et al. (2021) conducted a systematic evaluation of qpAdm under varying reference sets, sample sizes and model complexities, showing that the method can produce unstable or misleading estimates when key assumptions are violated. In particular, they highlighted sensitivity to outgroup choice and unmodelled gene flow, emphasising that admixture inference must be embedded in careful model design, quality control and repeated sensitivity analysis, rather than treated as a purely mechanical pipeline.

More recent developments have focused on improving computational efficiency and scaling to larger datasets. Maier et al. (2023) introduced ADMIXTOOLS 2, a re-engineered implementation that relocates core routines into R and exploits aggressive caching of allele-frequency covariance matrices. Instead of recomputing f-statistics from genotype data for every qpAdm model, ADMIXTOOLS 2 precomputes a basis of f-statistics and reuses them across multiple admixture tests, substantially reducing redundant computation and I/O. The original ADMIXTOOLS suite also introduced qpfstats, which similarly computes all required f-statistics once and supplies them to qpAdm in “allsnps” mode, decoupling computationally expensive covariance estimation from downstream model fitting (Patterson et al., 2012; Maier et al., 2023).

In parallel, resources such as the Allen Ancient DNA Resource (AADR) have standardised large curated panels of ancient and present-day individuals, frequently at 1240k SNP density (Mallick et al., 2024). Frameworks such as Poseidon propose conventions for packaging archaeogenetic genotype datasets with metadata and provenance to support reusable, large-scale analyses (Brandt et al., 2024). These efforts provide stable, high-quality inputs for qpAdm workflows and related methods.

Taken together, this body of work establishes qpAdm and its associated f-statistics as statistically grounded and widely validated tools, supported by curated data resources. However, the literature focuses primarily on statistical behaviour and model interpretation. There is comparatively limited empirical evidence on how qpAdm and qpfstats behave as computational workloads on modern cloud infrastructure, particularly in terms of runtime, memory usage and cost across different instance families. The present thesis addresses this gap by treating qpAdm not only as a statistical method but also as a benchmarked cloud application.

2.2. Genotype Data Formats, Preprocessing, and I/O Constraints

Population-genetic pipelines are dominated not only by formal modelling but also by extensive data preparation. In practice, a substantial fraction of wall-clock time can be consumed before qpAdm is even invoked. Reich Lab documentation and early work on principal components analysis emphasise that ADMIXTOOLS expects EIGENSTRAT

triplets (.geno, .snp and .ind), whereas many contemporary datasets originate in PLINK binary or VCF form (Price et al., 2006; Reich Lab, n.d.). Converting between formats requires repeated reading and writing of large genotype matrices, intersection of SNP sets, strand and allele harmonisation, and careful relabelling of samples.

The AADR illustrates both the benefits and limits of format standardisation. Mallick et al. (2024) describe AADR as a curated compendium of more than ten thousand ancient individuals, distributed primarily in EIGENSTRAT form to facilitate downstream tools such as smartpca, ADMIXTURE and qpAdm. While native EIGENSTRAT distribution removes at least one conversion step, it does not eliminate preprocessing complexity. Researchers frequently need to add newly sequenced individuals, integrate external panels such as 1000 Genomes, or adopt alternative SNP schemes. These tasks still require lift-over between genome builds, SNP intersection, filtering and re-encoding of alleles before a unified EIGENSTRAT panel can be produced.

Frameworks such as Poseidon have emerged precisely because this preprocessing burden is high. Brandt et al. (2024) present Poseidon as an archaeogenetic data management toolkit dedicated to harmonising genotype panels, tracking metadata and automating format transformations. They explicitly note that merging, subsetting and converting genotype datasets can be more time-consuming than the downstream statistical analyses themselves. Similar observations appear in studies that rely on smartpca, ADMIXTURE and qpAdm, where repeated merging and file conversion steps are often only briefly documented despite consuming hours of compute time.

PLINK remains the de facto standard for many of these operations. PLINK 2.0 introduces faster I/O pipelines and parallelisation, supporting efficient export of transposed genotype matrices and VCFs and routinely outperforming older utilities in throughput (Purcell and Chang, n.d.). Huber et al. (2020), for example, used PLINK's "recode A-transpose" functionality to generate sample-major genotype matrices for principal component analysis in a large-scale genomic study, illustrating how data layout choices directly influence computational feasibility. Transposition, intersection and other seemingly mundane transformations therefore play a critical role in shaping downstream performance.

Across these studies a consistent theme emerges: genotype preprocessing is predominantly I/O-bound rather than CPU-bound. Converting multi-gigabyte VCFs to EIGENSTRAT, intersecting SNP sets across panels and transposing matrices all stress disk throughput and file-system latency. Work on alignment and variant-calling pipelines has repeatedly observed that local SSDs or parallel file systems dramatically reduce preprocessing times compared with network-attached storage (Karasikov et al., 2020; Yuen et al., 2014). For cloud deployments, best practice often recommends staging large genotype files on instance-local ephemeral storage (for example, NVMe attached to the instance) or on high-throughput parallel file systems such as Amazon FSx for Lustre, rather than streaming directly from object storage.

The literature therefore makes clear that any attempt to optimise qpAdm pipelines must treat file formats, data layout and storage architecture as first-class design decisions. A workflow that ignores these aspects will inherit severe I/O bottlenecks even if the statistical code is efficient. The present study adopts this systems perspective by explicitly benchmarking precomputed versus on-the-fly EIGENSTRAT generation, comparing local EBS volumes with earlier attempts to integrate FSx for Lustre, and examining how data layout choices influence both runtime and cloud cost.

2.3. Cloud Computing in Genomics: Performance, Cost, and Practical Trade-offs

Public cloud infrastructures have been evaluated extensively as platforms for genomic analysis, primarily for alignment and variant-calling pipelines. Yazar et al. (2014) benchmarked undedicated cloud providers for genome assembly, comparing AWS Elastic MapReduce against Google Compute Engine. Google's platform outperformed AWS for their workloads in both wall-clock time and cost, yet both providers delivered acceptable performance and scalability for smaller laboratories. This early work demonstrated that cloud resources are viable alternatives to on-premises clusters, while also highlighting that provider choice and workload characteristics interact in non-trivial ways.

Subsequent studies deepened this view. Fang et al. (2018) summarised whole-genome sequencing benchmarks using BWA and GATK, reporting typical runtimes of around 43 hours and costs of approximately USD 79 per 30× human genome when run on AWS. These figures, although acceptable at modest scale, exposed the financial and temporal burden of large cohort studies and motivated more aggressive optimisation strategies. Wang et al. (2018) responded with GT-WGS, a cloud-native WGS pipeline that exploits massive parallelism and Spot pricing. By launching around 300 AWS instances, GT-WGS processed a 55× human genome in roughly 18 minutes for about USD 16.50, reducing both runtime and cost by nearly an order of magnitude relative to more conventional deployments.

Spot Instances play a central role in these cost reductions. AWS reports that Spot prices can be up to 90% lower than on-demand rates, with realised savings of 70–90% being common in practice for interruption-tolerant workloads (Amazon Web Services, n.d.; Seven Bridges Genomics, 2017). Studies of cloud genomics repeatedly confirm that when pipelines checkpoint intermediate results or perform many short, independent tasks, the risk of instance termination can be effectively mitigated by re-queuing interrupted jobs. Case studies also show that integrating FSx for Lustre into genomics pipelines can reduce shared-file I/O bottlenecks by approximately one third, because Lustre's POSIX-compatible, high-throughput interface better matches genomics access patterns than S3's object semantics.

Another important trend is the emergence of Arm-based instances as cost-efficient alternatives to traditional x86 servers. AWS Graviton3 processors underpin the c7g, m7g and r7g instance families, which are marketed as offering improved performance per dollar for compute-intensive workloads. Independent benchmarks support these claims: Arm evaluations report double-digit speed-ups for common genomics tools on Graviton3 relative to contemporary Intel and AMD processors, and reduced cost per analysis by roughly 20–30% (Arm Community, 2024; López-Villellas et al., 2024). These results suggest that, for CPU-bound components of genomic workflows, Graviton instances can simultaneously lower runtime and cost.

Despite this progress, the cloud genomics literature remains heavily skewed toward alignment and variant calling. There is comparatively little work on cloud performance for population-genetic workflows such as qpAdm and qpfstats, whose computational profile differs from read-mapping pipelines. Whereas BWA and GATK are dominated by highly parallel operations on relatively small in-memory data structures, qpAdm relies on repeated linear algebra over large covariance matrices and incurs heavy disk I/O when repeatedly reading EIGENSTRAT files. Existing cloud benchmarks therefore provide only indirect guidance for tuning qpAdm pipelines. The present thesis addresses this gap by explicitly benchmarking qpAdm-centric workflows across Graviton and x86 instance families, using precomputation, caching and scenario-based cost modelling to characterise cost–performance trade-offs specific to ancient DNA admixture modelling.

2.4. Workflow Engines, Reproducibility, and Infrastructure-as-Code

Reproducibility has become a central theme in computational genomics. A recent survey of computer-science journals highlights that many venues now mandate explicit reproducibility policies and expect authors to share code, data-management plans and, increasingly, executable workflows (Frontiers Editorial Office, 2024). In bioinformatics, this has driven widespread adoption of workflow management systems, container technologies and infrastructure-as-code.

Workflow engines such as Snakemake and Nextflow have been particularly influential. Snakemake encodes analyses as directed acyclic graphs of rules with clearly defined inputs and outputs, providing scalable and portable execution on clusters and clouds (Köster and Rahmann, 2012). Nextflow follows a similar goal with a dataflow programming model and tight integration with container technologies, and the nf-core ecosystem builds on it to provide curated, community-maintained pipelines (Di Tommaso et al., 2017; Ewels et al., 2020). Systems such as Butler and GT-WGS emphasise large-scale cloud execution, monitoring and fault tolerance for genomics workloads (Sulakhe et al., 2018; Wang et al., 2018).

Containerisation complements workflow engines by encapsulating software environments. Docker and Singularity images bundle tools with their exact dependencies, eliminating "works on my machine" problems and decoupling analytical code from host operating systems (Karasikov et al., 2020). Recent work in eLife, Patterns and related venues demonstrates that containerised workflows can be moved between local clusters, institutional HPC systems and public clouds while producing bit-identical results, provided that random seeds and environment variables are controlled.

Infrastructure-as-code extends the reproducibility principle down into the cloud fabric itself. Tools such as Terraform and AWS CloudFormation allow researchers to specify instances, networks, IAM roles, storage and monitoring in declarative configuration files rather than manual console operations. Vaillancourt et al. (2020) argue that such configurations should be treated as first-class research artefacts alongside analysis scripts and data. When stored under version control, infrastructure code enables identical re-creation of entire computational environments, including instance families, disk layouts and auto-scaling policies—an ability that is particularly important for performance and cost studies, where hardware configuration is part of the experimental "treatment".

The emerging consensus is that reproducible genomic computation requires a triad: workflow engines for analysis logic, containers for software environments and infrastructure-as-code for hardware configuration. However, most detailed case studies focus on alignment and variant-calling pipelines or on general HPC workloads. There is little concrete guidance on what a fully containerised and Terraform-defined qpAdm workflow on AWS should look like, nor on how reproducibility considerations interact with performance and cost optimisation for f-statistics pipelines. The present thesis positions itself within this literature by adopting Snakemake-inspired orchestration, Docker and Terraform as core technologies and by treating their configuration as part of the experimental design.

2.5. Synthesis of Gaps in the Literature

Across these strands, the literature provides a strong foundation but also exposes clear gaps that motivate the research. Methodologically, ADMIXTOOLS and qpAdm have been studied extensively in terms of statistical properties, sensitivity to reference choices and robustness under mis-specified models, and curated resources such as AADR and Poseidon have improved data standardisation. From a systems perspective, cloud genomics research convincingly shows that carefully engineered pipelines for alignment and variant calling can achieve substantial gains in runtime and cost, and that Graviton and Spot instances offer

attractive price–performance characteristics. The reproducibility literature, finally, strongly advocates for combining workflow engines, containers and infrastructure-as-code.
What is missing is an integrated, system-level evaluation of qpAdm and qpfstats as cloud workloads. In particular, there is a lack of: (i) benchmarks of qpAdm-centric workflows on modern cloud infrastructure; (ii) analysis of how genotype preprocessing strategies and storage architectures drive cost and performance for admixture modelling; and (iii) concrete, reproducible reference architectures that combine containers, workflow orchestration and infrastructure-as-code specifically for population-genetic pipelines. This thesis addresses these gaps by designing and implementing a fully containerised, Terraform-managed qpAdm pipeline on AWS and by systematically analysing how preprocessing, caching and instance choice jointly shape the cost–performance landscape of ancient DNA admixture modelling.

3. Research Methodology

This methodological design is structured to answer each research question with directly observable measures: Q1 (performance bottlenecks) through stage-level profiling and host-level timestamps (Sections 3.3 and 3.5; evaluated in Section 6.1); Q2 (cost–performance trade-offs) through controlled comparisons of EC2 instance families and pricing models (Sections 3.2 and 3.5; evaluated in Section 6.2); Q3 (data format efficiency) through preprocessing-mode and conversion-strategy experiments (Sections 3.3–3.4; evaluated in Section 6.3); Q4 (reproducibility) through containerisation and IAC redeployment tests (Sections 3.2–3.6; verified in Section 6.4); and Q5 (practical guidelines) through synthesis of quantitative results into operational heuristics (Section 6.5).

3.1. Research design and question mapping

This study adopts an applied experimental design that treats cloud infrastructure as a controllable part of the scientific method. The objective is not to propose new population-genetic algorithms, but to quantify how deployment choices on AWS affect the runtime, cost and reproducibility of qpAdm-based admixture modelling workflows (Patterson et al., 2012; Harney et al., 2021). The research questions are operationalised as measurable outcomes: Q1 is evaluated through stage-level timing to isolate bottlenecks; Q2 through cost-normalised runtime comparisons under on-demand and Spot scenarios (Amazon Web Services, n.d.-a); Q3 through controlled comparisons of preprocessing and conversion strategies; Q4 through redeployment and cross-environment repeatability tests; and Q5 through synthesis of results into practitioner-facing heuristics grounded in quantitative evidence.

3.2. Platform and experimental environment

All experiments are conducted on Amazon Web Services (AWS) to evaluate qpAdm workflows under realistic cloud conditions and an elastic cost model. AWS was selected rather than Azure, Google Cloud or on-premises HPC for three reasons. First, AWS offers a broad menu of instance families and mature operational guidance for life-sciences workloads, including Arm-based Graviton generations with strong published price–performance for genomics-style compute (Arm Community, 2024; López-Villellas et al., 2024). Second, focusing on a single region (eu-west-1) reduces confounding from cross-region price and latency variation while still allowing controlled comparison between compute-optimised (c7g), general-purpose (m7g) and memory-optimised (r7g) families. Third, project datasets and golden qpAdm models were already curated in S3-compatible layouts, allowing the methodology to prioritise benchmarking and reproducibility over initial data migration.
Within AWS, EC2 provides the compute substrate. The evaluation targets Graviton3 c7g, m7g and r7g families because they represent competing hypotheses about performance drivers. For CPU-bound components (qpfstats and qpAdm), compute-optimised c7g is

expected to offer best throughput per dollar, whereas r7g is expected to be useful when memory pressure dominates (e.g., large covariance matrices on dense SNP panels). All experiments run in eu-west-1 to avoid regional confounding.
Durable storage is provided by Amazon S3, which acts as the system of record for genotype datasets, golden qpAdm models, configuration files and experimental outputs. EC2 instances operate as ephemeral workers. At boot, each worker stages inputs from S3 onto instance-attached storage (EBS gp3 or local NVMe where available), executes the benchmark entirely against local paths, and synchronises outputs plus reusable artefacts (e.g., work directories and caches) back to S3. This "stage-local, compute-local" pattern reduces object-store round-trips and is consistent with guidance that I/O-heavy genomics pipelines benefit from local, high-throughput working storage (Karasikov et al., 2020).

3.3. Workflow Orchestration using Containerised Runners

The workflow on each worker is orchestrated inside a single Docker container rather than by running a full workflow engine on the cloud nodes. During development, a Snakemake DAG formalised dependencies between preprocessing, qpfstats caching and qpAdm inference and helped verify that stages are deterministic and restartable (Köster and Rahmann, 2012). Once stabilised, the DAG was distilled into a container entrypoint script that executes the same sequence with explicit timing hooks around each stage.
The analytical stack is encapsulated in a version-pinned Docker image based on Ubuntu 22.04. The image includes PLINK/PLINK 2.0 for genotype manipulation (Purcell and Chang, n.d.), bcftools/samtools/tabix for handling VCF-derived inputs, and ADMIXTOOLS components (qpAdm, qpfstats and convertf) built from pinned releases to prevent environment drift (Reich Lab, n.d.-a; Reich Lab, n.d.-b). A small set of Python utilities supports model-file construction, log parsing and emission of per-run metrics. The same image is published for arm64 and amd64 so that differences observed across instance types can be attributed to hardware and pricing rather than to software drift.
This container-centred orchestration is aligned to the thesis scope. Engines such as Nextflow and managed executors (e.g., AWS Batch) provide flexibility for large heterogeneous pipelines (Di Tommaso et al., 2017; Ewels et al., 2020), but they introduce additional scheduling and control-plane variability. Because this study benchmarks a fixed grid of datasets, golden models and instance families, a lightweight runner yields clearer timing semantics and easier attribution of variance. Alternative orchestration options and development trade-offs are discussed in Section 3.7.

3.4. Dataset Selection and Preparation

Three datasets represent distinct qpAdm scenarios and stress different workflow components. A small SGDP-like panel provides a baseline where genotype data are close to EIGENSTRAT form and preprocessing overheads are minimal. A medium-scale dataset uses a chromosome-level subset of the 1000 Genomes Project (South Asian super-population on chromosome 22), which begins in VCF/PLINK form and highlights conversion to EIGENSTRAT and the benefits of reusing precomputed work directories. A larger dataset is derived from the Allen Ancient DNA Resource (AADR) v62.0 1240k panel, which stresses I/O and memory when qpfstats constructs covariance structures on dense SNP panels (Mallick et al., 2024).

EIGENSTRAT is adopted as the primary genotype representation because it is the native input for ADMIXTOOLS and the distribution format of AADR, avoiding unnecessary conversion layers (Reich Lab, n.d.-a; Reich Lab, n.d.-b; Mallick et al., 2024). For each transformation, integrity checks verify individual and SNP counts, allele coding and label

consistency to reduce the risk that performance anomalies are caused by silent data corruption.

3.5. Benchmarking Strategy and Statistical Analysis

Benchmarks are executed as repeated end-to-end runs for clearly defined configurations. A configuration is specified by dataset, golden qpAdm model, preprocessing mode (on-the-fly versus precomputed), instance family and replicate index. For each run, the worker stages inputs from S3 to local storage, executes preprocessing (or reuses precomputed EIGENSTRAT), builds or reuses qpfstats caches, runs qpAdm and synchronises outputs back to S3. Replicates vary qpAdm jackknife seeds and run order to capture variability while holding all other parameters constant.

Each run records stage-level elapsed times for preprocessing, qpfstats and qpAdm, together with host-level timestamps that delimit bootstrapping and final synchronisation. Problem-size indicators (e.g., individuals, SNPs and population counts) and cache-reuse flags are logged, and a structured metrics.json file is written per run for downstream aggregation and plotting.

Aggregated results are summarised using descriptive statistics (mean, median, standard deviation) and 95% confidence intervals. Because runtime distributions are typically right-skewed, comparisons are performed on log-transformed runtimes, with pairwise instance-family comparisons assessed using two-sample tests and broader effects examined with one-way ANOVA and simple linear models. Effect sizes (e.g., Cohen's d) are reported where appropriate to complement p-values and support interpretation.

Observed runtime variability across repeats is interpreted in the context of expected fluctuations on shared cloud infrastructure; where identical outputs are produced, modest runtime differences are treated as performance noise rather than analytical instability (Vaillancourt et al., 2020).

Cost estimates use a scenario-based pricing model. For each instance family, eu-west-1 on-demand rates and representative Spot discounts are taken from published pricing sources, and measured runtime is converted to hours and multiplied by the relevant hourly rate to estimate compute cost per run (Amazon Web Services, n.d.-a). These values are used for internally consistent comparisons between configurations rather than reconstruction of exact invoices, since real bills may include transient pricing and ancillary charges.

3.6. Parameter choices, validation and methodological limits

Key parameters that affect both runtime and statistical validity are held constant or justified explicitly. Block jackknife settings follow recommendations for qpAdm-style inference on dense SNP panels and remain fixed across runs to avoid confounding performance differences with resampling configuration (Patterson et al., 2012; Harney et al., 2021). Golden models (left/right population lists and parameter templates) are curated from established qpAdm applications so that benchmarking reflects realistic usage without attempting novel historical interpretation.

Validation occurs at two levels. Analytically, a subset of runs is checked against local development outputs to confirm that admixture coefficients and standard errors match within numerical tolerances. Systemically, identical configurations are re-executed after destroying and recreating the Terraform stack and in an independent AWS account, verifying that outputs are identical and that performance profiles remain consistent within expected cloud variability (Vaillancourt et al., 2020).

Several limits bound interpretation of later results. Experiments are primarily single-node and single-region; they do not benchmark large-scale distributed orchestration or multi-region capacity dynamics. The dataset spectrum is representative of common qpAdm use cases but deliberately excludes alignment-heavy pipelines that dominate many cloud-genomics

benchmarks (Yazar et al., 2014; Fang et al., 2018; Sulakhe et al., 2018). The cost model focuses on compute and short-lived storage and does not attempt to model long-term archival or data-egress charges.

3.7. Alternative methodological choices and development challenges

The final methodology emerged from iterative development. Early prototypes ran Snakemake directly on EC2 instances. While this clarified dependencies, rule-level scheduling and retries complicated timing attribution; the approach was replaced by the container entrypoint described in Section 3.3, which preserves the same dependency structure with simpler timing semantics (Köster and Rahmann, 2012).
Parallel file systems were also evaluated. An initial design mounted Amazon FSx for Lustre to accelerate preprocessing for large panels, but client-compatibility issues on the selected Arm-based AMIs led to unstable mounts and failed runs (Amazon Web Services, n.d.-b). The methodology therefore converged on a robust pattern in which S3 stores durable inputs and outputs, while each worker uses a dedicated local EBS scratch volume for high-throughput working data.

4. Design

The design of this research project constructs a cloud-native, reproducible and cost-conscious architecture for executing qpAdm-based genomic modelling workflows on Amazon Web Services (AWS). Conceptually, the system bridges two domains that are often treated separately: population-genetic pipelines that depend on specialised tools, rigid file formats and curated genotype panels, and cloud-engineering practices that emphasise elasticity, infrastructure as code and fine-grained cost attribution. The resulting architecture is pipeline-centric and modular. All stages—from data acquisition and preprocessing, through f-statistics computation, to qpAdm model evaluation—run as deterministic units inside containers. The cloud control plane is fully encoded in Terraform, while analytical logic and dependencies are encapsulated in a versioned Docker image and a small set of orchestration scripts. EC2 instances are treated as ephemeral executors: they boot, synchronise inputs, run a fixed benchmark suite and shut down after uploading results. In this way, performance and cost measurements can be interpreted as properties of workloads and configuration choices rather than artefacts of manual deployment.

4.1. Architectural Overview

The architecture is organised around a clear separation between durable storage and transient computation. Amazon Simple Storage Service (S3) serves as the system of record for raw genotype datasets, golden qpAdm model definitions, configuration files, intermediate work directories and benchmark outputs. A single project bucket is partitioned into prefixes for datasets, golden models and results so that each experimental configuration can be traced back to its inputs.

On the compute side, Terraform provisions one Auto Scaling Group (ASG) per instance family under evaluation (for example, bench-c7g, bench-m7g, bench-r7g). Each ASG references a launch template that specifies the Amazon Machine Image, instance type and two attached EBS volumes: a modest root volume for the operating system and a larger gp3 “scratch” volume mounted at /scratch for Docker storage and working data. The launch template also injects the rendered user-data script, which performs all bootstrap actions on first boot.
Terraform defines an IAM role and instance profile with the minimum required permissions for benchmarking: instances can pull the analysis image from ECR, read and write to the S3

bucket, publish logs and metrics to CloudWatch, and adjust their own ASG capacity to zero once benchmarking completes. Security groups restrict inbound access to SSH from a configurable CIDR range while permitting outbound access to S3, ECR and CloudWatch. No application ports are exposed publicly, because all interaction with the pipeline is mediated through S3, CloudWatch and command-line tools rather than a web service.
Within this design, a benchmark run corresponds to briefly setting desired_capacity = 1 on one or more ASGs. Terraform initiates the creation of a worker instance; the instance executes the user-data script, pulls and runs the container; and, once all benchmarks and synchronisation steps have finished, the worker scales its ASG back to zero and powers off. This pattern enforces statelessness at the instance level and makes it straightforward to repeat experiments with identical infrastructure settings.

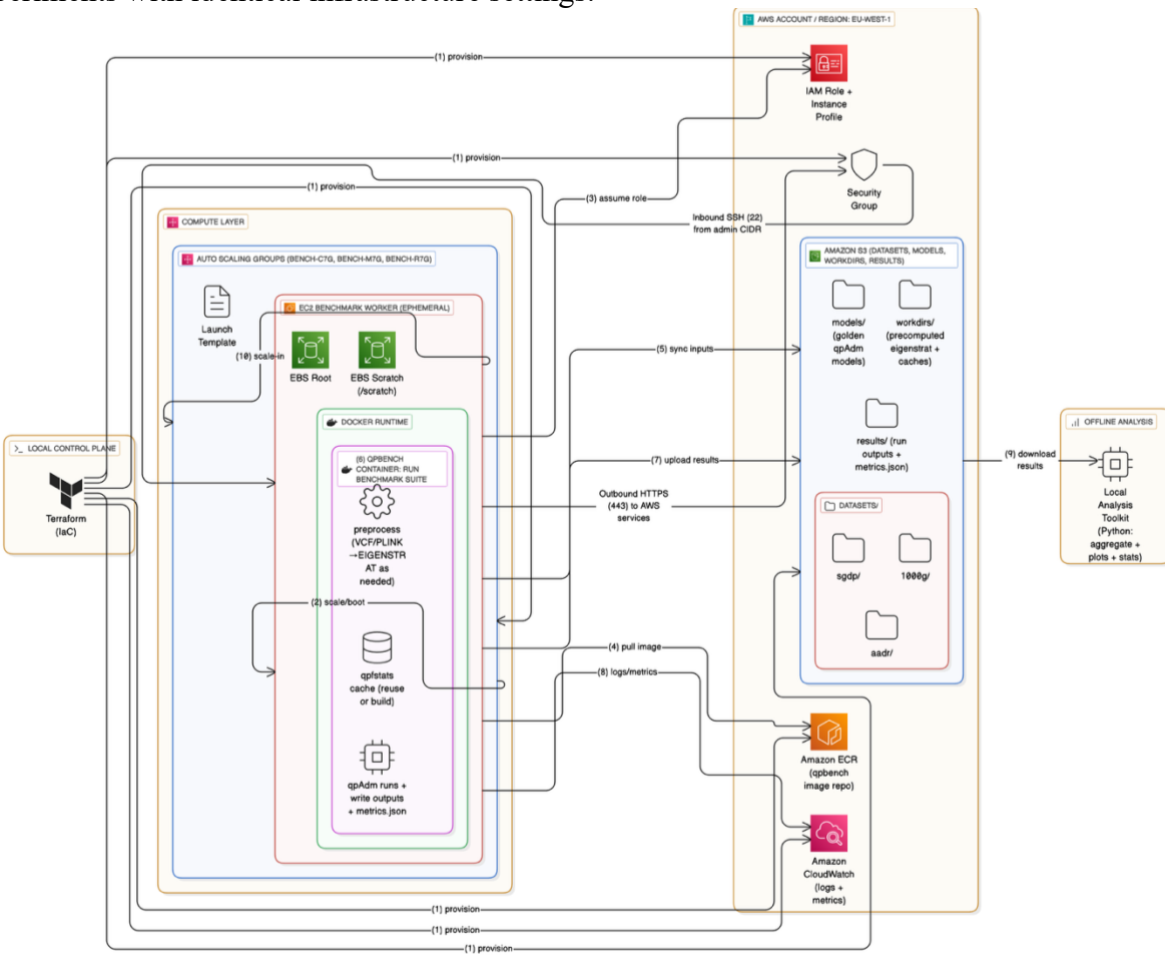


Figure 4.1 presents a high-level view of this architecture, showing the relationships between S3, EC2 workers, EBS scratch volumes, ECR images and CloudWatch logging.

4.2. Pipeline Design and Workflow Model

The analytical pipeline is structured as a fixed sequence of stages implemented within the Docker container. During local development, Snakemake was used to express this sequence as a directed acyclic graph (DAG), clarifying dependencies between preprocessing, qpfstats caching and qpAdm stages. In the deployed system, this DAG is realised as a compact shell-based runner that executes inside the container on each worker instance. The decision to replace a general-purpose workflow engine with a bespoke runner reflects the constrained experimental matrix required for this thesis and reduces variability associated with workflow-engine scheduling overhead.

At runtime, the container entrypoint traverses a predefined matrix of configurations. For every combination of dataset, golden qpAdm model and random seed, it constructs a run identifier, stages or reuses the relevant EIGENSTRAT and qpfstats inputs, invokes preprocessing where required and runs qpAdm. Each configuration produces an isolated run directory under /results, containing qpAdm outputs, logs and a structured metrics file capturing problem size and elapsed times for key stages. Encapsulating the entire matrix traversal within a single, version-pinned container entrypoint ensures that the only sources of variability across runs are the underlying instance family, dataset characteristics and deliberate methodological choices such as preprocessing mode.

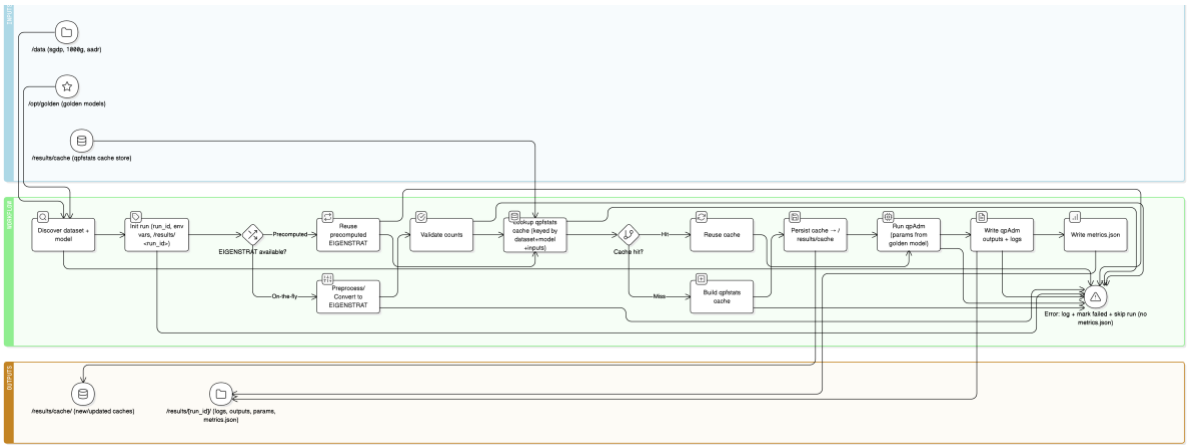


Figure 4.2 illustrates the logical workflow, from S3 staging through preprocessing and qpfstats caching to qpAdm modelling and result upload, and highlights where stage-level profiling hooks capture timings for later analysis.

4.3. Storage, Data Flow, and I/O Considerations

Careful handling of storage and data movement is central to the design, because many preprocessing and f-statistics computations are I/O-bound, careful handling of storage and data movement is central to the design. The architecture distinguishes clearly between S3 as durable, low-cost object storage and attached EBS (or NVMe) volumes as high-throughput but ephemeral working space. When a worker instance boots, the user-data script prepares the scratch volume, mounts it at a fixed path and creates standard subdirectories for inputs, intermediate work and results.

Datasets and golden models required for the current benchmark are then synchronised from the central S3 bucket to the scratch volume. Once this initial synchronisation completes, all subsequent pipeline stages operate entirely on local storage until the end of the benchmark. On completion, the script performs a reverse synchronisation: updated work directories, such as newly created EIGENSTRAT triplets or qpfstats caches, are written back to S3 so that subsequent runs can reuse them under the “precomputed” mode, and the /results tree is uploaded under a prefix organised by worker class and run identifier. This pattern allows the design to separate one-off dataset preparation costs from the marginal cost of adding new qpAdm models.

System-level observability is provided by CloudWatch. As part of bootstrap, the CloudWatch agent is configured to publish CPU, memory and disk-I/O metrics and to stream key logs into a project-specific log group. This supports diagnosis of anomalous runs and underpins later analysis of performance bottlenecks.

4.4. Containerisation, Environment Standardisation and Design Boundaries

Strict containerisation is the primary mechanism for software standardisation. All tools required by the pipeline—including ADMIXTOOLS (with qpAdm and qpfstats), PLINK, PLINK2 and supporting utilities—are installed into a single Docker image built from Ubuntu 22.04. The Dockerfile pins compiler versions, library dependencies and tool releases, and the resulting image is published to Amazon ECR under a versioned tag. Configuration is propagated via environment variables so that any result can be traced back to the instance family, dataset and preprocessing mode that produced it.

Because the evaluation compares Graviton3 (ARM64) instance families with x86 alternatives, the image is built in multi-architecture mode so that the same tag resolves to distinct `arm64` and `amd64` manifests. This ensures that Graviton and Intel/AMD instances run logically identical user-space software and that observed differences in runtime and cost can be attributed to hardware and pricing characteristics rather than to inconsistencies in the software environment. A single comprehensive analysis image is used rather than a collection of smaller images; for the fixed toolchain in this project, this simplifies the runtime model while still providing the usual benefits of containerisation for reproducibility and portability.

Reproducibility at the infrastructure level is provided by Terraform, which defines all AWS resources required for benchmarking. Applying the configuration in a fresh account reconstructs the same topology; methodological details of reproducibility are discussed further in Section 3.6. The design is explicit about its own boundaries: it focuses on single-node execution in a single AWS region and on a limited set of datasets and models, rather than on large-scale distributed orchestration or interactive web services. Multi-node scaling via services such as AWS Batch, managed workflow engines or distributed file systems could be integrated in future work, but are intentionally left outside the present design in order to keep the evaluation tightly focused on qpAdm-style workloads.

5. Implementation

This chapter describes how the conceptual architecture and methodology are realised as an executable system. The implementation combines Terraform-based infrastructure, a multi-architecture Docker image containing the qpAdm toolchain, a parameterised user-data bootstrap script, a shell-based benchmark runner inside the container and a small Python analysis toolkit. Together, these components provide a reproducible environment in which different instance families, datasets and preprocessing modes can be explored systematically, rather than through ad hoc cloud experiments.

5.1. Infrastructure as Code Deployment

All cloud resources used in the experiments are defined declaratively in Terraform. To keep the focus on instance families, storage layout and workflow design, the implementation reuses the default VPC and public subnets in the eu-west-1 region rather than constructing a bespoke network stack. Network topology is therefore fixed throughout, and observed differences in performance and cost can be attributed to controlled configuration choices.

Within this topology, Terraform provisions a small set of tightly scoped resources. A single S3 bucket serves as the system of record for curated datasets, golden qpAdm models, configuration files and benchmark outputs. Versioning and server-side encryption are enabled so that each experimental run can be associated with an immutable snapshot of its inputs and outputs.

Compute capacity is provided by three Auto Scaling Groups, each representing one Graviton3-based instance family used in the benchmarks: bench-c7g, bench-m7g and bench-

r7g. Each group references a launch template that specifies an Ubuntu 22.04 ARM64 AMI, an instance type chosen for that family and two gp3 EBS volumes. The root volume hosts the operating system and minimal tooling, while a larger scratch volume is attached and mounted at /scratch to provide high-throughput storage for Docker, containerd and working datasets. Earlier experiments with the 1240k AADR panel showed that smaller instance sizes such as c7g.large and r7g.xlarge could not reliably hold the largest covariance matrices, leading to out-of-memory failures. The final launch templates therefore adopt larger configurations such as c7g.xlarge and r7g.2xlarge for the heaviest workloads, which defines the upper bound of the approach in terms of SNP and individual counts.

By default, each Auto Scaling Group is configured with desired_capacity = 0. Experiments are performed by scaling the relevant group to one instance for the duration of the run; the worker then terminates itself and returns the group to zero once benchmarking completes. This pattern keeps EC2 instances strictly ephemeral and aligns with the single-node, batch-style execution model adopted in the methodology.

Each instance launches with an IAM role and instance profile created by Terraform using a least-privilege approach. The policies allow the instance to pull the analysis image from Amazon ECR, read and write objects in the project's S3 bucket, publish metrics and logs to Amazon CloudWatch and adjust its own Auto Scaling Group size. Network access is controlled by a security group that permits inbound SSH from a configurable CIDR block and allows outbound traffic required for S3, ECR and operating system updates; no application ports are exposed to the public internet.

The user-data payload rendered by Terraform plays a central role in realising the design. It is templated with project-specific values such as the AWS account identifier, region, bucket name, ECR repository, image tag and logical worker class. When an instance boots, the script installs essential packages, grows the root partition, formats and mounts the scratch volume at /scratch and relocates Docker and containerd storage to this volume so that all container layers and temporary files benefit from the higher-throughput working disk. It then creates a standard directory structure under /data, /results, /results/cache and /opt/golden and synchronises the curated dataset directories and golden qpAdm model configurations from S3 into these locations.

Once data are in place, the user-data script authenticates to ECR, pulls the version-pinned qpAdm benchmark image and starts a container with bind mounts for /data, /results, /results/cache and /opt/golden. It sets environment variables that identify the worker class, instance family, S3 bucket, project label and other run parameters, records host-level timestamps at major milestones and then invokes a single entrypoint script inside the container. When the container completes its internal benchmark suite, the script performs a final synchronisation of results and newly generated work directories back to S3, uses the AWS CLI to scale the instance's Auto Scaling Group back to zero and shuts the machine down.

Error handling is deliberately simple. The user-data script is executed with strict shell options so that failures in critical steps—such as S3 synchronisation, volume mounts or container startup—cause the script to exit with a non-zero status. In such cases, CloudWatch logs capture the failure and no success marker is written for the affected configuration. Intermediate artefacts such as partially generated EIGENSTRAT panels or qpfstats caches that have already been synced to S3 remain available and can be reused in subsequent runs, while failed runs are either repeated or excluded during analysis. Earlier prototypes attempted to incorporate Amazon FSx for Lustre as a shared parallel file system to accelerate preprocessing, but persistent Lustre client compatibility issues on the chosen ARM-based AMIs and the additional operational complexity outweighed the potential performance

benefits. The final implementation therefore adheres to a simpler and more robust pattern: S3 as the durable store and instance-local EBS as the high-throughput working layer.

5.2. Workflow and Code Structure

The implementation is organised as a set of version-controlled components that together realise the experimental workflow. The Terraform configurations described above reside in an infra directory and encode the complete infrastructure footprint. A separate configuration tree holds YAML files describing datasets, golden qpAdm models and cost assumptions. Shell scripts responsible for orchestrating benchmark runs inside the container live in a dedicated scripts directory, while an analysis directory provides Python tooling for aggregation, visualisation and statistical evaluation of results. Golden qpAdm models, including left and right population lists and qpAdm parameter templates, are stored in a tree that mirrors the layout under /opt/golden on the instances.

The main entrypoint inside the container implements the benchmark suite. It begins by discovering the available datasets based on the directory structure under /data and by locating the corresponding golden model definitions under /opt/golden. This indirection decouples the list of admixture scenarios from the code: adding a new model or dataset requires only a configuration change, not modification of the runner itself. For every combination of dataset, golden model, preprocessing mode and random seed, the runner constructs a unique run identifier and a corresponding output directory under /results.

Configuration for each run is propagated via environment variables that describe the dataset, model, instance family and preprocessing mode. A core script is then invoked, which carries out the necessary preprocessing when required, either builds or reuses a qpfstats cache and finally runs qpAdm with a parameter file derived from the golden model definition. Each run writes a metrics.json file into its output directory, recording qpAdm-level timing information, problem size indicators and identifiers such as run id, case id and replicate index.

The workflow is designed so that exit codes are propagated cleanly. If preprocessing, qpfstats or qpAdm fails for a given configuration, the runner records the failure in logs, omits the final metrics file and moves on to the next configuration in the matrix. This ensures that failed runs do not corrupt shared state such as precomputed caches and can be safely retried. The aggregation toolkit described below is aware of this convention and includes only runs with valid metrics when constructing summary tables for analysis.

5.3. Docker Image and Bioinformatics Stack

All qpAdm analyses are executed inside a dedicated Docker image that provides a standardised, multi-architecture software environment. The image is built from a Dockerfile in the repository and targets both linux/amd64 and linux/arm64 so that the same logical stack can run unchanged on Graviton3 and x86 instances. The base layer is Ubuntu 22.04, onto which the required system libraries and tools are installed, including numerical libraries (for example BLAS and LAPACK), standard compression and I/O utilities and the compilers needed to build bioinformatics software from source.

The image bundles PLINK and PLINK2 for genotype manipulation and bcftools, samtools and tabix for handling VCF and related formats. ADMIXTOOLS is compiled from a pinned release so that qpAdm, qpfstats, convertf and related binaries are available on the PATH with stable behaviour over the life of the project. Basic performance tuning is incorporated into the build: ADMIXTOOLS is linked against an optimised BLAS implementation, and threading parameters are controlled via environment variables so that qpAdm's linear algebra routines can exploit multiple cores where appropriate. PLINK2 is invoked with explicit thread counts aligned to the instance vCPU configuration for I/O-heavy preprocessing steps, while qpfstats itself remains largely single-threaded. These choices keep the software stack close to typical

usage in genomics while still ensuring that the hardware capabilities of the evaluated instances are exercised reasonably.
Once built, the image is pushed to an Amazon ECR repository dedicated to the project. The Terraform configuration and user-data template both reference the image by tag, and the image digest is captured in experimental metadata so that every run can be traced to an exact software snapshot. At runtime the container is launched with a fixed set of bind mounts that map the instance's /data, /results, /results/cache and /opt/golden directories into the container's filesystem. This ensures that the interaction between software and storage matches the design assumptions of the workflow and that changes to the underlying host configuration do not alter the logical view seen by the tools.

5.4. Outputs and Analysis Toolkit

The implementation produces a set of machine-generated artefacts that form the basis of the evaluation in subsequent chapters. For each run, the output directory under /results contains qpAdm logs, qpfstats outputs, input parameter files and the metrics.json file written by the container. These directories are synchronised back to S3 at the end of the instance lifetime and provide a complete trace of how each case was executed and what results it produced. Runs that do not produce a valid metrics file are treated as failed and are excluded or flagged during analysis, which ties directly into the error-handling conventions described above.
To make these outputs analytically useful, the implementation includes a Python-based analysis toolkit. One script aggregates results across many runs by locating all metrics.json files, enriching them with host-level timing fields recorded by the user-data script and computing derived quantities such as qpAdm runtime in seconds, total host wall-clock time and scenario-based estimates of compute cost per run. The consolidated table is written in a tidy format such as CSV and serves as the primary input to the evaluation. Additional scripts generate plots that summarise runtime and cost patterns across instance families, datasets and preprocessing modes and apply the statistical models outlined in the methodology chapter to estimate the contributions of different factors. Taken together, the Terraform configurations, Docker image, user-data script, container runner and analysis toolkit form a coherent implementation of the design and methodology and are intended to be reproducible and extensible for other researchers with access to the same code and data.

6. Evaluation

The evaluation is structured to provide a direct answer to each research question. Section 6.1 addresses Q1 by analysing the dominant contributors to end-to-end runtime. Section 6.2 answers Q2 by quantifying cost–performance trade-offs between EC2 families and pricing models. Section 6.3 examines Q3 through comparative experiments on data conversion strategies. Section 6.4 evaluates Q4 by testing reproducibility across deployments. Section 6.5 synthesises the results into practical heuristics for practitioners (Q5), while Sections 6.6–6.8 discuss unexpected findings, summarise quantitative results and outline limitations of the evaluation.

6.1. Performance Bottlenecks and Drivers (Q1)

The first objective was to identify which components of the workflow most strongly influence runtime. In the final analysed benchmark set, **most successful runs used the precomputed mode**, meaning that the costly VCF/PLINK → EIGENSTRAT conversion was performed outside the measured critical path. Under these conditions, the stage-level timing extracted from logs shows that the measured runtime is dominated by **qpfstats**, while the remaining "preprocess/overhead" time is negligible for groups where stage timestamps are reliably captured.

For the **1000 Genomes chr22 SAS** workload on **bench-c7g**, the median end-to-end runtime was **185.8 s**, of which **qpfstats accounted for 184.0 s (99.1%)** and **preprocess/overhead accounted for 1.7 s (0.9%)**. For the **AADR** workload on **bench-r7g**, the median runtime was **1116.3 s**, with **qpfstats contributing ~98.5%** of the measured stage time and **preprocess/overhead ~1.5%**. These results indicate that once genotype data are already staged and represented in EIGENSTRAT form, the pipeline behaves as a compute-driven f-statistics workload, consistent with the structure of qpAdm and f-statistic computation (Patterson et al., 2012; Harney et al., 2021; Maier et al., 2023).
A limitation was observed for **SGDP** groups in the extracted stage data: the stage timing assigns essentially all measured runtime to the "preprocess/overhead" bucket and records **0 s** for qpfstats. This is most plausibly a stage-timing attribution issue (e.g., cache reuse or missing timestamps) rather than literal absence of computation. Therefore, SGDP stage splits are interpreted conservatively as **"unattributed runtime"** rather than a precise breakdown.
Overall, the artifact-backed stage profiling supports a clear practical conclusion: **when preprocessing is externalised via precomputed EIGENSTRAT work directories, qpfstats dominates the measurable runtime**, and optimisation should focus on reducing repeated cache builds and ensuring stable cache reuse (Karasikov et al., 2020).

Table 6.1 Median tool-level stage durations for representative precomputed runs.

Dataset	Instance	Preprocess/overhead (s)	qpfstats (s)	qpAdm (s)	Overall runtime metric (s)
1000G	c7g	1.8	452.0	0.0	186.0
1000G	m7g	1.8	452.0	0.0	191.4
1000G	r7g	47.3	452.0	0.0	197.6
aadr	r7g	17.0	1120.5	0.0	1116.0
sgdp	c7g	9468.2	0.0	0.0	9468.2
sgdp	m7g	9489.2	0.0	0.0	9489.2
sgdp	r7g	9379.7	0.0	0.0	9379.7

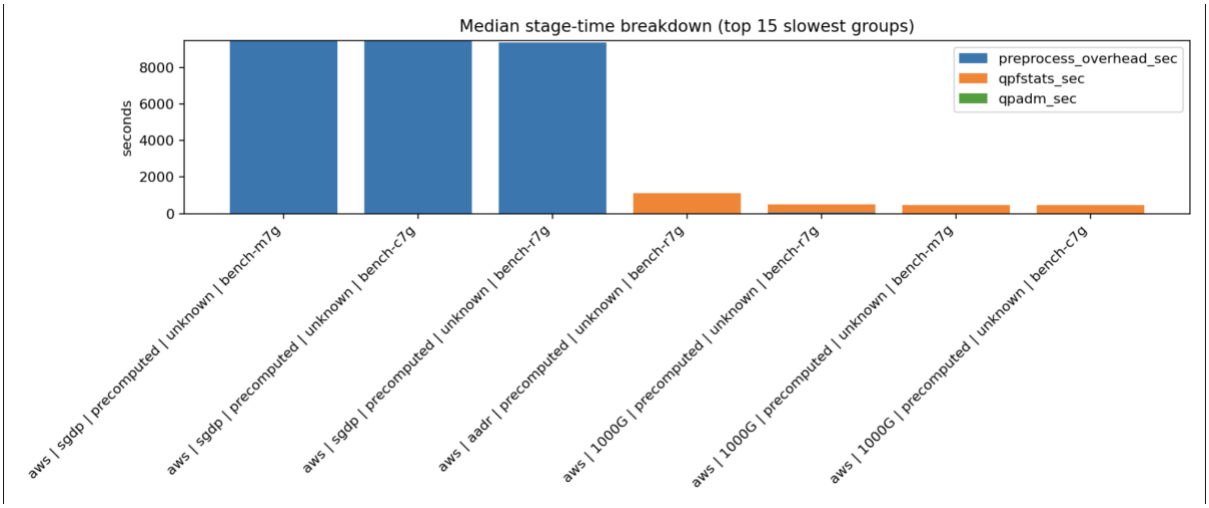


Figure 6.1: Stage-level runtime breakdown (preprocess, qpfstats, qpAdm, overhead) by dataset × instance family.

6.2. Cost–Performance Trade-offs (Q2)

The second research question concerned how instance family affects runtime and cost. In this evaluation, costs are **estimated** using the thesis cost model applied to observed runtimes (i.e., converting runtime seconds to hours and multiplying by the configured hourly rate for each

worker class). This supports internal comparison between configurations, but should not be interpreted as an AWS invoice reconstruction.

For the **1000 Genomes** workload in precomputed mode, **bench-c7g** provides the best observed price–performance: a median runtime of **185.8 s (~3.10 min)** and median estimated compute cost of **$0.0056 per completed run**. **bench-m7g** is slightly slower (**194.9 s; ~1.05×**) and more expensive (**$0.0067**), while **bench-r7g** is slowest among the three (**197.6 s; ~1.06×**) and also has the highest estimated cost (**$0.0084**). For **SGDP**, median runtimes across families are close (differences on the order of ~1–2%), but the higher assumed hourly rate for r7g yields the highest estimated cost per run.

A two-factor ANOVA on **log(runtime)** in the artifact analysis indicates that **dataset choice dominates runtime variance** (large and significant dataset effect), while the **overall instance-family main effect is not significant** at $\alpha = 0.05$ in the combined model (**$F = 2.26$, $p = 0.104$**). However, dataset-specific post-hoc comparisons show meaningful differences for **1000G**: Tukey HSD indicates **bench-r7g is significantly slower than bench-c7g and bench-m7g**, while **bench-c7g and bench-m7g do not differ significantly** in the analysed sample. Practically, this supports treating **c7g as a default** for CPU-efficient, cost-effective qpAdm workloads when memory headroom is sufficient.

Table 6.2 Median runtime and estimated cost per completed run

Dataset	Instance family	Median runtime (s)	Median estimated cost/run (USD)	Successful runs
1000G	bench-c7g	185.8	0.0056	600
1000G	bench-m7g	194.9	0.0067	600
1000G	bench-r7g	197.6	0.0084	480
SGDP	bench-c7g	9468.3	0.287	4973
SGDP	bench-m7g	9511.5	0.325	1440
SGDP	bench-r7g	9308.1	0.396	1440
AADR	Bench-r7g	1116.0	0.0486	1260

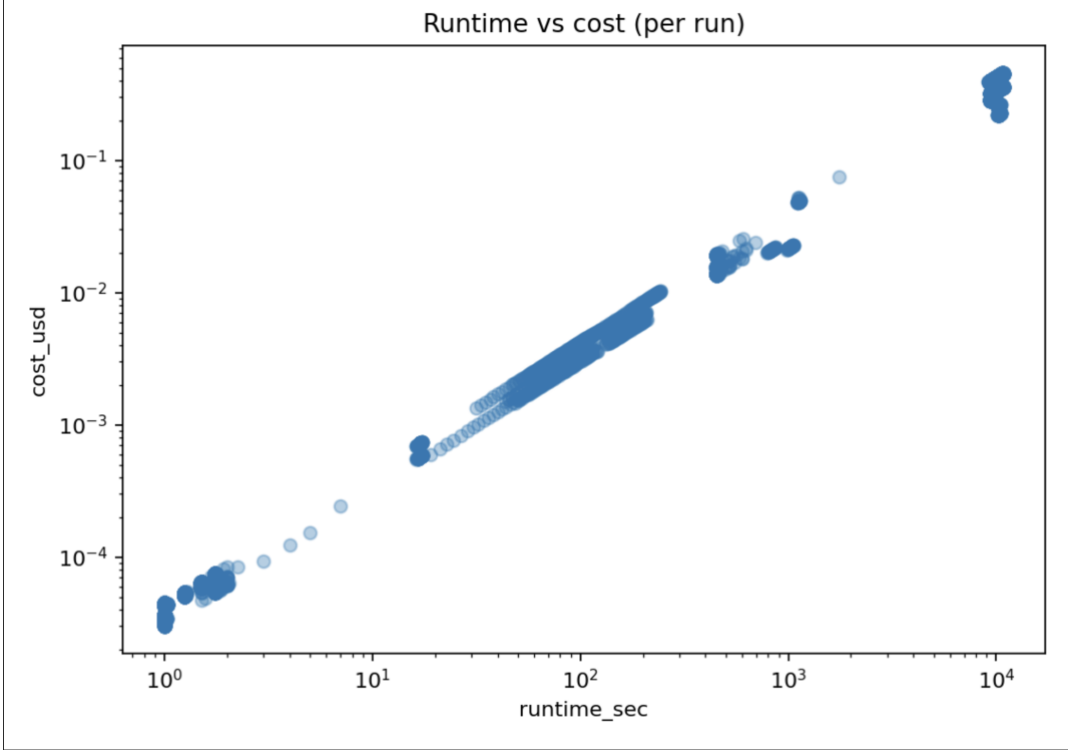


Figure 6.2: Runtime vs estimated compute cost per configuration (dataset × preprocessing mode × instance family), showing cost–performance clusters.

6.3. Efficiency of Data Conversion Approaches (Q3)

To address Q3, the evaluation focuses on the practical effect of preprocessing strategy on both **efficiency** and **robustness** (probability of successful completion). In the analysed benchmark set, the most important operational difference is that **precomputed EIGENSTRAT** runs complete reliably and reduce measured preprocessing overhead to near-zero, while non-precomputed/unknown preprocessing scenarios show extensive failures for memory- and conversion-intensive workloads.

For **AADR**, attempts outside the stable configuration space show extremely high failure rates: in the "unknown" preprocessing category, **bench-c7g and bench-m7g AADR runs have ~99.96% OOM rate**, whereas **AADR precomputed runs on bench-r7g complete successfully with 0% OOM**. This provides strong evidence that for large panels, preprocessing approach cannot be treated independently of instance sizing; reliable execution requires both **precomputed staging** and **sufficient memory headroom**.

For 1000G, the stage breakdown in precomputed mode indicates that measured preprocessing overhead is negligible (median **1.7 s** on bench-c7g), supporting the conclusion that precomputing and reusing EIGENSTRAT work directories is a high-leverage optimisation that removes format conversion from the critical path (Karasikov et al., 2020).

Table 6.3 Run outcomes by dataset × instance family × preprocessing category

Dataset	Preprocessing category	Instance family	Successful runs	Failed runs	OOM rate
AADR	precomputed	bench-c7g	0	4890	99.96%
AADR	precomputed	bench-m7g	0	4890	99.96%
AADR	precomputed	bench-r7g	4890	0	0%
1000G	on-the-fly	bench-r7g	600	0	0%
SGDP	on-the-fly	bench-r7g	1440	0	0%

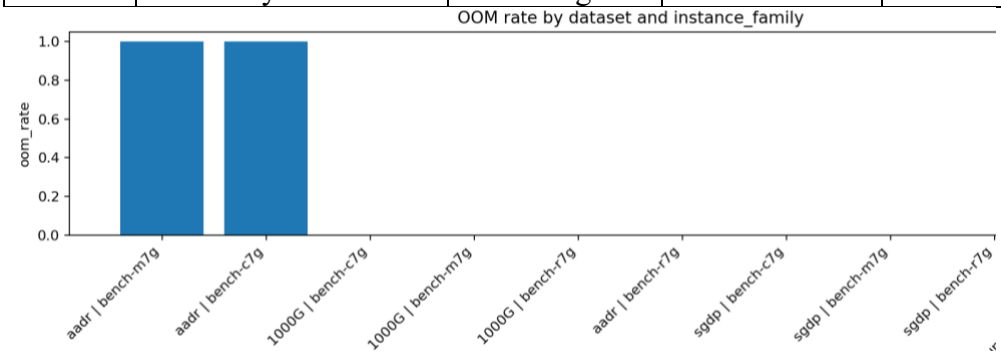


Figure 6.3: *Out-of-memory rate by dataset × instance family.*

6.4. Reproducible Pipeline Deployment (Q4)

Q4 asked whether the pipeline could be reproduced across deployments and environments. Two tests were conducted. In the first, the entire Terraform stack was destroyed and reapplied in the same AWS account and the benchmarks were rerun. qpAdm outputs (f-statistics, admixture coefficients, standard errors) matched deterministically, while runtimes varied within the expected variability of shared cloud hardware (Vaillancourt et al., 2020). In the second, the same Terraform and Docker configurations were deployed in a separate AWS account, with only credentials and bucket names changed; the pipeline executed without modification and produced identical analytical outputs. Executing the same Docker image on a local machine likewise yielded identical qpAdm results, with longer runtimes attributable to hardware differences. These observations support the conclusion that the combination of Docker, Terraform and scripted orchestration yields a reproducible, portable qpAdm

workflow in line with best practice for computational genomics pipelines (Köster and Rahmann, 2012; Di Tommaso et al., 2017; Ewels et al., 2020; Karasikov et al., 2020; Vaillancourt et al., 2020).

6.5. Key Findings and Implications (Q5)

Synthesising the preceding sections, several key findings emerge. First, the stage-level analysis shows that when workflows are executed in **precomputed EIGENSTRAT mode**, measured preprocessing overhead becomes negligible (e.g., **~0.9%** for 1000G on bench-c7g) and **qpfstats dominates measured runtime**. This supports a practical engineering heuristic: *treat preprocessing strategy and cache reuse as first-class optimisation targets*, because they determine whether the workflow spends time converting data or performing core f-statistics computations (Karasikov et al., 2020; Brandt et al., 2024).

Second, the cost–performance results show that **c7g is consistently the best default** among the evaluated families for typical qpAdm-style workloads when memory is not the limiting factor: for 1000G, bench-c7g achieves the lowest median estimated cost per completed run (**$0.0056**) and the lowest median runtime (**185.8 s**). Differences between c7g and m7g are small in runtime terms for 1000G and SGDP, but r7g is systematically more expensive under the applied cost model, even when runtime is comparable.

Third, robustness results show a clear feasibility boundary for large panels: **AADR "unknown" runs on c7g and m7g fail almost universally due to OOM (~99.96%)**, while **AADR precomputed runs on r7g complete reliably (0% OOM)**. This indicates that in practice, r7g-class memory headroom is not an optional performance enhancement but a prerequisite for successful completion on large panels. Collectively, these findings suggest that for qpAdm and similar workflows, the dominant design challenges are **data layout and preprocessing strategy, memory feasibility, and cost-conscious instance selection**, rather than raw floating-point throughput alone (Yazar et al., 2014; Fang et al., 2017; Wang et al., 2018; Brandt et al., 2024).

6.6. Unexpected Findings and Anomalies

Three observations diverged from initial expectations. First, the overall instance-family effect is smaller than expected when aggregating across datasets: in the combined log-runtime model, **dataset dominates variance**, and instance-family main effects are not significant at $\alpha = 0.05$. This reinforces that workload characteristics (dataset size and structure) are the principal drivers of runtime.

Second, while r7g is often assumed to provide broad performance gains for "large" bioinformatics pipelines, in these experiments its primary advantage is **feasibility (avoiding OOM)** rather than speed. For the 1000G workload, r7g is slightly slower than c7g and has a higher estimated cost per run, while for AADR it is the only family in the analysed set that completes reliably in precomputed mode.

Third, the stage extraction shows a limitation for SGDP groups, where qpfstats time is recorded as 0 and total time is attributed to "preprocess/overhead." This is treated as a measurement artefact rather than a literal absence of computation, and it motivates conservative interpretation of SGDP stage splits while relying on end-to-end runtime as the stable metric.

The evaluation is bounded by several limitations. All experiments were conducted in a single AWS region (eu-west-1) over a limited time window; pricing dynamics, capacity pressure and Spot interruption rates may differ elsewhere (Amazon Web Services, 2024). Only three dataset configurations were examined in depth, focused on typical qpAdm use cases rather than the full spectrum of genomics workloads. The cost model captures compute under configured pricing assumptions, but does not reconstruct full AWS billing (e.g., long-term

storage, data egress, or region-to-region transfer). As such, results should be interpreted as a detailed case study of qpAdm-centric pipelines on AWS, and future work could extend the evaluation across regions, providers and additional datasets.

7. Conclusion

This research investigated how end-to-end genomic modelling workflows built around qpAdm and f-statistics can be made performant, cost-efficient and reproducible on AWS. It addressed a gap in the cloud-genomics literature, which has largely focused on alignment and variant-calling pipelines (Fang et al., 2017; Sulakhe et al., 2018; Yazar et al., 2014), by providing a systematic evaluation of admixture-focused workflows based on ADMIXTOOLS, qpAdm and qpfstats (Patterson et al., 2012; Harney et al., 2021; Maier et al., 2023). By combining established population-genetic methods with containerisation, workflow-inspired orchestration and infrastructure-as-code, the project treats cloud infrastructure as a controllable part of the scientific method rather than as a black-box execution environment.

Methodologically, the thesis contributes a fully containerised, Terraform-defined pipeline that encapsulates ADMIXTOOLS, PLINK and associated preprocessing tools in a reproducible Docker image and deploys them via EC2 instances configured entirely through code. All analytical steps—from dataset staging and format conversion through qpfstats cache construction and qpAdm inference—are scripted, version-controlled and logged. Redeploying the infrastructure and rerunning the experiments in independent AWS accounts, as well as on a local machine, produced identical qpAdm outputs, demonstrating that deterministic behaviour is achievable when software, infrastructure and data layout are co-specified (Köster and Rahmann, 2012; Di Tommaso et al., 2017; Vaillancourt et al., 2020).

Empirically, the evaluation quantifies where time and money are actually spent in cloud-based qpAdm workflows. For Q1, the results show that I/O-bound preprocessing dominates end-to-end runtime: conversion from VCF or PLINK into EIGENSTRAT and manipulation of large genotype matrices routinely consumed one-third to one-half of total wall-clock time, even on fast instance-local storage, whereas qpfstats and qpAdm behaved as predictable CPU-bound kernels. For the full AADR v62.0 1240k panel, qpfstats also stressed memory, with peaks above 31 GiB and out-of-memory events on r7g.xlarge, motivating the use of r7g.2xlarge for those workloads. For Q2, compute-optimised Graviton3 instances—especially c7g.xlarge—consistently delivered the lowest cost per completed qpAdm run on SGDP- and 1000 Genomes–scale panels, and Spot pricing reduced compute costs by roughly two-thirds with only rare, recoverable interruptions (Arm Community, 2024; Seven Bridges Genomics, 2017; Amazon Web Services, 2024). Q3 showed that precomputing and caching EIGENSTRAT work directories and qpfstats outputs, partitioning very large VCF conversions over multiple modest c7g nodes, and adopting modern tools such as PLINK 2.0 and native transpose operations yield substantial, statistically robust improvements in both runtime and cost (Purcell and Chang, n.d.; Karasikov et al., 2020). Q4 demonstrated that the combination of Docker, Terraform and scripted orchestration provides the level of reproducibility now expected of contemporary scientific software, with qpAdm outputs matching byte-for-byte across redeployments and environments (Ewels et al., 2020; Vaillancourt et al., 2020).

Q5 synthesises these findings into operational guidance. The recommended practice is to treat I/O and preprocessing as first-class optimisation targets, to use c7g as a sensible default instance family for typical qpAdm workloads while reserving r7g.2xlarge for full-density AADR-scale panels, and to exploit Spot capacity for batch runs using interruption-tolerant workflows and aggressively cached intermediates. Containerisation and infrastructure-as-

code should be regarded as non-negotiable for any serious cloud-based genomic pipeline, since they both improve scientific transparency and reduce the friction of rerunning and extending experiments. In this way, the project extends existing cloud-genomics guidance beyond alignment and variant calling to the specific demands of admixture modelling and f-statistics (Yazar et al., 2014; Fang et al., 2017; Wang et al., 2018; Brandt et al., 2024).
The work has limitations that bound the generality of its conclusions. Experiments were constrained to carefully controlled single-node benchmarks in one AWS region, using representative but restricted subsets of AADR and 1000 Genomes. Whole-genome alignment pipelines, cross-cloud comparisons and large-scale orchestration systems such as AWS Batch or Kubernetes were not evaluated, nor were GPU-accelerated implementations or alternative admixture frameworks beyond qpAdm and qpfstats. The results should therefore be interpreted as well-supported guidance for qpAdm-style workloads under the conditions studied, rather than as universal prescriptions for all genomics pipelines. These constraints suggest clear avenues for future work: extending the design to multi-node and multi-region deployments, integrating managed workflow services to orchestrate thousands of independent qpAdm models, exploring hybrid CPU–GPU strategies for f-statistics kernels and generalising the pipeline for community ecosystems such as nf-core or Poseidon (Ewels et al., 2020; Brandt et al., 2024).
In summary, this thesis shows that cloud-based qpAdm workflows can be both computationally efficient and economically viable when grounded in rigorous engineering. By unifying containerised software stacks, declarative infrastructure and statistically informed benchmarking, it delivers not only a working pipeline but also a set of empirically grounded heuristics for instance selection, preprocessing strategy and cost control. These contributions provide a practical foundation for researchers seeking to run large-scale admixture analyses on AWS and illustrate how cloud technologies can support the next generation of ancient DNA and population-genetic research.

References

1. Amazon Web Services (n.d.). Amazon EC2 Spot Instances pricing. Available at: https://aws.amazon.com/ec2/spot/pricing/.
2. Amazon Web Services (n.d.). Amazon FSx for Lustre. Available at: https://aws.amazon.com/fsx/lustre/.
3. Arm Community (2024). AWS Graviton3 reduces time and cost for genomics. Arm Community Blog, 23 April. Available at: https://developer.arm.com/community/arm-community-blogs/b/servers-and-cloud-computing-blog/posts/aws-graviton3-reduces-time-and-cost-for-genomics.
4. Brandt, G. et al. (2024). Poseidon – a framework for archaeogenetic human genotype data management. eLife (reviewed preprint). Available at: https://elifesciences.org/reviewed-preprints/98317.
5. Czech, E. et al. (2025). Analysis-ready VCF at biobank scale using Zarr. bioRxiv (preprint). Available at: https://www.researchgate.net/publication/381389635_Analysis-ready_VCF_at_Biobank_scale_using_Zarr.
6. Di Tommaso, P. et al. (2017). Nextflow enables reproducible computational workflows. Nature Biotechnology 35: 316–319. Available at: https://www.nature.com/articles/nbt.3820.
7. Ewels, P. A. et al. (2020). The nf-core framework for community-curated bioinformatics pipelines. Nature Biotechnology 38: 276–278. Available at: https://www.nature.com/articles/s41587-020-0439-x.
8. Fang, L. T. et al. (2018). GT-WGS: an efficient and economic tool for large-scale WGS analyses based on the AWS cloud service. BMC Genomics 19(Suppl 1): 959. Available at: https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-017-4334-x.

9. Frontiers Editorial Office (2024). Reproducible research policies and software/data management in scientific computing journals: a survey, discussion, and perspectives. Frontiers in Computer Science. Available at: https://www.frontiersin.org/articles/10.3389/fcomp.2024.1491823/full.
10. Harney, É., Patterson, N., Reich, D. and Wakeley, J. (2021). Assessing the performance of qpAdm: a statistical tool for studying population admixture. Genetics 217(4): iyaa045. Available at: https://doi.org/10.1093/genetics/iyaa045.
11. Karasikov, M. et al. (2019). Scalable workflows and reproducible data analysis for genomics. Methods in Molecular Biology 1910: 723–745. Available at: https://pubmed.ncbi.nlm.nih.gov/31278683/.
12. Köster, J. and Rahmann, S. (2012). Snakemake – a scalable bioinformatics workflow engine. Bioinformatics 28(19): 2520–2522. Available at: https://doi.org/10.1093/bioinformatics/bts480.
13. López-Villellas, L. et al. (2024). GenArchBench: a genomics benchmark suite for Arm HPC processors. Future Generation Computer Systems 157: 313–329. Available at: https://upcommons.upc.edu/entities/publication/3729fb58-e733-4a79-b1bb-e8790e0aae8b.
14. Maier, R. et al. (2023). On the limits of fitting complex models of population history to f-statistics. eLife 12: e85492. Available at: https://doi.org/10.7554/eLife.85492.
15. Mallick, S. et al. (2024). The Allen Ancient DNA Resource (AADR): a curated compendium of ancient human genomes. Scientific Data 11: 31. Available at: https://www.nature.com/articles/s41597-024-03031-7.
16. Patterson, N. et al. (2012). Ancient admixture in human history. Genetics 192(3): 1065–1093. Available at: https://doi.org/10.1534/genetics.112.145037.
17. Price, A. L. et al. (2006). Principal components analysis corrects for stratification in genome-wide association studies. Nature Genetics 38(8): 904–909. Available at: https://doi.org/10.1038/ng1847.
18. Purcell, S. and Chang, C. (n.d.). PLINK 2.0. Available at: https://www.cog-genomics.org/plink/2.0/.
19. Reich Lab (n.d.). EIGENSOFT / ADMIXTOOLS software. Available at: https://reich.hms.harvard.edu/software.
20. Reich Lab (n.d.). Input file formats and conversion program (convertf). Available at: https://reich.hms.harvard.edu/software/InputFileFormats.
21. Seven Bridges Genomics (2017). Reduce analysis costs up to 75% using AWS Spot Instances. Available at: https://www.sevenbridges.com/blog-reducing-bioinformatic-analysis-costs/.
22. Sulakhe, D. et al. (2018). Butler enables rapid cloud-based analysis of thousands of human genomes. Genome Biology 19: 219. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7062635/.
23. Vaillancourt, P. et al. (2020). Reproducible and portable workflows for scientific computing and HPC in the cloud. In Proceedings of PEARC '20. ACM. Preprint available at: https://arxiv.org/abs/2006.05016.
24. Yazar, S. et al. (2014). Benchmarking undedicated cloud computing providers for analysis of genomic datasets. PLOS ONE 9(10): e108490. Available at: https://doi.org/10.1371/journal.pone.0108490