GenoMAS: A Multi-Agent Framework for Scientific Discovery via Code-Driven Gene Expression Analysis

Haoyang Liu[1], Yijiang Li[2], and Haohan Wang[1]

[1]University of Illinois at Urbana-Champaign
[2]University of California, San Diego
{hl57, haohanw}@illinois.edu, yijiangli@ucsd.edu

Abstract

Gene expression analysis holds the key to many biomedical discoveries, yet extracting insights from raw transcriptomic data remains formidable due to the complexity of multiple large, semi-structured files and the need for extensive domain expertise. Current automation approaches are often limited by either inflexible workflows that break down in edge cases or by fully autonomous agents that lack the necessary precision for rigorous scientific inquiry. **GenoMAS** charts a different course by presenting a team of LLM-based scientists that integrates the reliability of structured workflows with the adaptability of autonomous agents. **GenoMAS** orchestrates six specialized LLM agents through typed message-passing protocols, each contributing complementary strengths to a shared analytic canvas. At the heart of **GenoMAS** lies a guided-planning framework: programming agents unfold high-level task guidelines into Action Units and, at each juncture, elect to advance, revise, bypass, or backtrack, thereby maintaining logical coherence while bending gracefully to the idiosyncrasies of genomic data.

On the GenoTEX benchmark, GenoMAS reaches a Composite Similarity Correlation of 89.13% for data preprocessing and an F_1 of 60.48% for gene identification, surpassing the best prior art by 10.61% and 16.85% respectively. Beyond metrics, GenoMAS surfaces biologically plausible gene–phenotype associations corroborated by the literature, all while adjusting for latent confounders. Code is available at https://github.com/Liu-Hy/GenoMAS.

1 Introduction

Scientific research increasingly depends on complex computational analysis, yet the design and execution of such analysis remain labor-intensive, error-prone, and difficult to scale. From genomics [23, 103, 125], materials discovery [25], drug development [49, 106], and epidemiology [69, 9], to climate modeling [60] and personalized medicine [36], the analytical backbone of modern science involves highly structured, multi-step workflows written in code. These workflows must integrate raw or semi-structured data, apply statistical or mechanistic models, and yield interpretable results—all under the constraints of evolving domain knowledge, platform variation, and methodological rigor. Recent advances in large language models (LLMs) have led to rapid progress in general-purpose agents capable of decomposing tasks [118, 113, 79, 18], interacting with tools [86, 138, 147, 49], and producing executable code [30, 126, 167, 114, 140]. These developments have raised the prospect that LLM-based agents might soon contribute meaningfully to scientific discovery by automating analysis pipelines, exploring hypotheses, or refining computational models [8, 60]. However, realizing this promise requires bridging a fundamental gap between general reasoning ability and the structured, precision-driven nature of scientific computation.

While many LLM-based agents have shown competence in retrieving documents, calling APIs, or planning abstract tasks [165, 154, 46, 124], these capabilities fall short in domains where scientific progress depends on code. In fields such as transcriptomics [90, 69, 9], protein engineering [106], and statistical genetics [36, 76], research workflows are encoded as sequences of programmatic transformations, each tailored to the idiosyncrasies of a specific dataset, model assumption, or experimental design. Analysts routinely write custom scripts to handle malformed metadata, adjust for confounding variables, map deprecated identifiers, and reimplement statistical procedures in response to batch

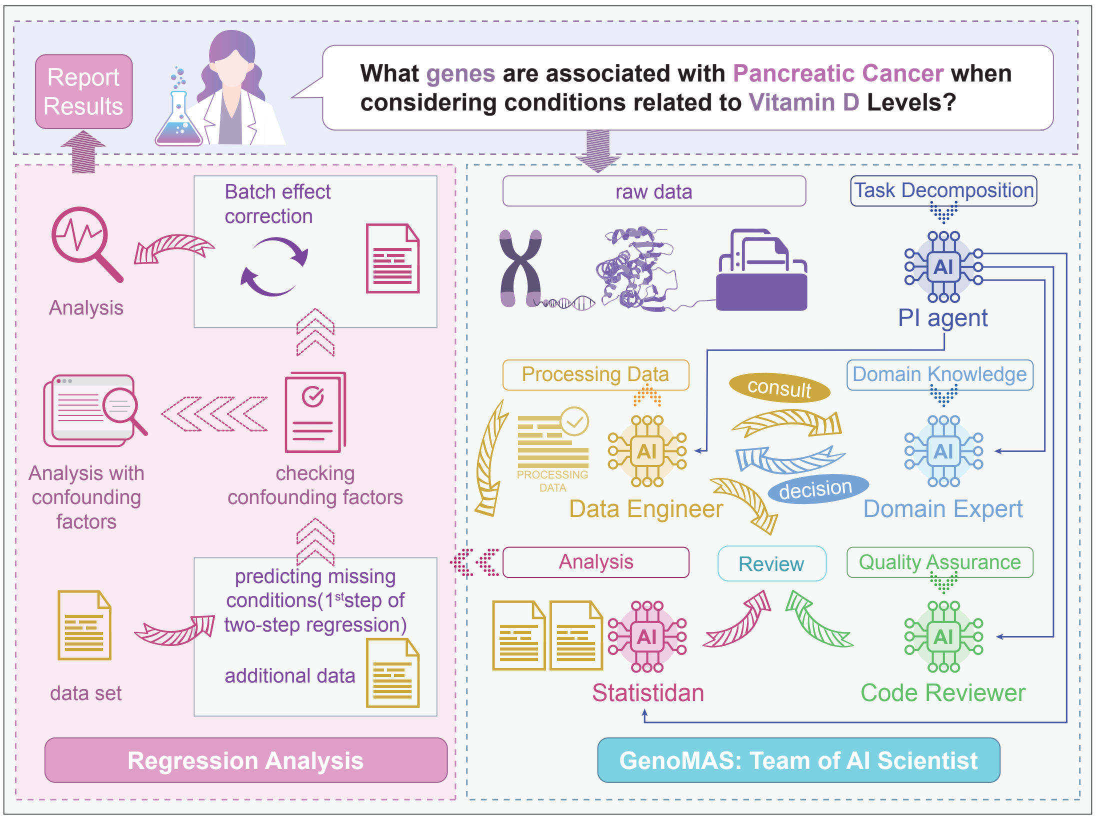


Figure 1: Multi-agent collaboration in our GenoMAS method.

effects or platform noise [51, 54]. These operations cannot be abstracted into declarative graphs or offloaded to black-box tools; they require direct, editable code. Even in large biomedical consortia with standardized pipelines, deviations and adaptations remain common [23, 103, 22]. Scientific automation, therefore, hinges not merely on planning workflows, but on writing, revising, and validating the code that executes them.

Despite growing interest in agentic AI [119, 65], current systems rarely address this requirement. Most multi-agent frameworks operate by composing tool calls [86, 138, 49], ranking retrieved functions [147], or coordinating modular invocations across structured graphs [165, 154, 46]. Some support dynamic routing [68, 92], temperature tuning [46], or neural module selection [153], but even these operate within fixed architectural templates. A few systems attempt code-level control [154, 46], but lack structured revision mechanisms, validation layers, or domain-aware correction. Critically, these frameworks are typically benchmarked on synthetic tasks [146, 49], where correctness is judged by output format or tool compatibility—not by scientific validity. In complex domains like gene expression analysis, even small errors in preprocessing or model selection can invalidate results entirely [35, 14]. Yet current agent systems offer no reliable means to detect, debug, or recover from such errors once code is executed. As a result, they remain unsuitable for scientific workflows where correctness is inseparable from interpretability and domain context.

In this paper, to bridge the gap between general-purpose agentic reasoning and the domain-specific precision required for scientific computation, we present **GenoMAS** (**Geno**mic data analysis through LLM-based **M**ulti-**A**gent **S**ystem), a framework that reframes scientific agents not only as workflow orchestrators, but also as collaborative programmers (Figure 1). Instead of invoking tools or chaining templates, GenoMAS agents generate, revise, and validate executable code tailored to each scientific task. Applied to gene expression analysis, the system automates complex workflows that previously required domain experts to script by hand. The system is evaluated on GenoTEX [66], a benchmark reflecting the demands of end-to-end scientific coding and thus provides a focused testbed for evaluating code-generating agents, curating a suite of 1,384 gene–trait association tasks spanning heterogeneous cohorts, evolving metadata, and statistical confounders. GenoMAS achieves state-of-the-art performance while operating entirely from raw data inputs. These results suggest that effective scientific automation requires agents that not only reason and retrieve, but that code.

Our contributions are as follows:

- We introduce **GenoMAS**, a multi-agent framework for scientific automation that treats agents as collaborative programmers rather than tool orchestrators, enabling end-to-end code generation for complex genomic analysis tasks.
- We develop a guided planning mechanism that encodes workflows as editable action units—structured textual directives that can be transformed into, and refined as, executable code—balancing precise control with autonomous error handling.
- We demonstrate that GenoMAS supports heterogeneous agent composition, allowing distinct LLMs with varying strengths (e.g., code synthesis, language reasoning, scientific review) to operate in coordinated roles within the same execution loop.

2 Related Work

LLM-based Agents The emergence of LLMs has enabled the development of autonomous agents capable of complex reasoning and task execution [118, 79, 112, 113, 149, 137]. These agents leverage LLMs as their cognitive core, augmenting basic language capabilities with structured reasoning approaches and external tool use [115, 120, 41, 144]. Early agent methods explored decomposing complex tasks into manageable sub-goals [128, 161, 33, 115, 77] and executing them sequentially. More sophisticated agents organize reasoning into tree [145, 41] or graph structures [10], enabling exploration of multiple solution paths. Critical to agent performance are mechanisms for self-reflection and iterative refinement [134, 74, 123, 19, 140], consistency checking [120], and integration with external tools and knowledge bases [64, 160, 86, 37, 85], which promises to transform LLMs from passive text generators into active problem-solving agents.

Multi-Agent System Given the capabilities of LLMs, multi-agent collaboration is expected to further enhance problem-solving performance [126, 107, 30, 117, 159]. In such systems, agents adopt specialized roles (i.e., role-playing) coordinated through structured protocols [141, 29, 152]. For example, some agentic methods [44, 84, 29] organize agents into different roles emulating human collaboration for software development. Complementary strategies, such as goal decomposition and task planning [128, 161, 33, 77, 18], and feedback mechanisms [47, 139, 37, 150, 155], have also shown effectiveness. Beyond performance, recent work explores sociocognitive dynamics in multi-agent systems, revealing emergent social behaviors and theory-of-mind-like reasoning in simulated environments [97, 105, 163, 59, 83]. More recent works have explored failure modes [15, 156] and training frameworks [72, 148] for multi-agent systems.

LLM Agents for Scientific Discovery One of the most ambitious applications of LLM agents lies in scientific research, where they are being developed to assist—or even automate—various stages of the discovery process [71, 99, 96, 8, 60, 149], including peer review [164, 130, 94, 53]. Recent systems demonstrate autonomous hypothesis generation [20], research assistance [95], and materials discovery [50]. For instance, AI Scientist [71] demonstrates that frontier LLMs can independently complete an entire research cycle, while ResearchAgent [8] generates research problems, methodologies, and experimental designs, refining them iteratively using scientific literature. Recent efforts have also integrated LLMs into domain-specific inquiries in mathematics [89, 109, 7, 28], physics [73, 56, 157], chemistry [102, 101, 11, 38, 168], biology [136, 75, 162, 135, 48], and medicine [100, 142, 67], either through prompting or fine-tuning models on specialized datasets.

Positioning of Our Work Existing agentic systems for scientific discovery typically operate at the level of task planning, document retrieval, or modular tool orchestration. While recent frameworks have demonstrated capabilities in hypothesis generation [8], experimental design [50], or literature-driven reasoning [53, 164], they rarely address settings where agents must write and revise executable code under scientific constraints. GenoMAS targets this gap directly: it treats scientific automation as a coding problem, not a retrieval or coordination problem.

3 Preliminaries

3.1 Gene Expression Data Analysis

Gene expression data analysis is a cornerstone of computational biology, offering critical insights into the molecular mechanisms that govern cellular function and disease [90, 14, 70]. Advances in high-throughput technologies—such as microarrays [1] and RNA sequencing [125, 103]—have enabled the generation of vast transcriptomic datasets, now housed in large-scale repositories including the Gene Expression Omnibus [22, 31] and The Cancer Genome Atlas [111, 129]. This work centers on gene-trait association (GTA) analysis, a key task that identifies genes whose expression patterns correlate with specific phenotypic traits while rigorously controlling for confounding biological variables [35, 133].

3.2 Problem Formulation

We address two classes of GTA problems following the GenoTEX benchmark [66]:

- **Unconditional GTA:** Given gene expression data $X \in \mathbb{R}^{n \times p}$ where n is the number of samples and p is the number of genes, and trait values $y \in \mathbb{R}^n$ (binary or continuous), identify the set of genes $G^* \subseteq \{1, ..., p\}$ significantly associated with the trait.
- **Conditional GTA:** Given the above data and an additional condition variable $c \in \mathbb{R}^n$ (e.g., age, gender, or comorbidity), identify genes G_c^* significantly associated with the trait when accounting for the influence of this condition [55]. This conditional analysis helps explore direct gene-trait associations that exist independently of the condition's influence, enabling more precise insights for personalized medicine [116, 17, 39].

The standardized analysis pipeline consists of three main stages: (1) dataset selection from available cohort studies, (2) data preprocessing to extract and normalize gene expression and clinical features, and (3) statistical analysis using regularized regression models to identify significant genes [24].

3.3 Evaluation

We evaluate automated gene expression analysis methods using the GenoTEX benchmark [66], which provides standardized evaluation for each stage of the analysis pipeline described above. The framework assesses performance through three core tasks that mirror these stages:

Dataset Selection This task assesses a system's ability to identify and prioritize relevant datasets for analysis. Dataset Filtering (DF) measures binary classification accuracy in determining dataset relevance, while Dataset Selection (DS) evaluates whether the method selects the same dataset as domain experts when multiple candidates are available.

Data Preprocessing Preprocessing entails extracting gene expression data from different platform–microarray probes require mapping to gene symbols [54], and RNA-seq reads demand alignment and quantification [23, 58]. Gene identifiers must be normalized across evolving nomenclatures and database versions [93, 127, 12], and clinical metadata extracted from heterogeneous semi-structured formats [16]. Preprocessing quality is evaluated using three metrics:

- Attribute Jaccard (AJ) measures the overlap of extracted features (genes and clinical variables).
- Sample Jaccard (SJ) assesses the overlap of correctly integrated samples.
- Composite Similarity Correlation (CSC) = AJ $\times$ SJ $\times$ Corr_{avg}, where Corr_{avg} is the average Pearson correlation coefficient between common features (shared rows and columns) of the preprocessed and reference datasets, capturing both structural and numerical fidelity.

Statistical Analysis Gene identification employs interpretable models, predominantly Lasso regression [110] and its biologically informed variants [133, 143], which are well-suited for high-dimensional, sparse settings and yield compact, interpretable gene sets [35]. Critical steps includes batch effect correction (e.g., ComBat [51] and its extensions [158, 32]), adjustment for population stratification [151, 63], and imputation of missing values [62, 2]. Evaluation for Statistical Analysis primarily uses on AUROC [40], appropriate for the highly imbalanced setting (often <2% significant genes), and is complemented by Precision, Recall, F_1, and the Gene Set Enrichment Analysis (GSEA) Enrichment Score [104].

3.4 Technical Challenges

Automated GTA analysis presents a set of deeply intertwined challenges that extend beyond conventional machine learning tasks [3].

(i) High-dimensional sparsity: Gene expression datasets typically comprise over 20,000 genes measured across <1,000 samples, posing significant statistical hurdles [52], further exacerbated by biological noise and technical variability [122].

(ii) Platform heterogeneity: Distinct measurement technologies demand distinct pipelines—microarrays rely on probe-based hybridization with platform-specific mappings [1], while RNA-seq requires complex alignment and quantification workflows [103, 23].

(iii) Evolving gene nomenclature: The continuous evolution of gene names—marked by synonyms, deprecated identifiers, and context-specific aliases—necessitates robust normalization and disambiguation tools [93, 131, 127].

(iv) Heterogeneous metadata: Phenotypic information appears in varied formats, often requiring domain expertise to extract standardized variables from free-text descriptions or infer information from indirect sources [21].

(v) Confounding factors: Batch effects [57, 122], population stratification [151], and hidden covariates [34, 42] can introduce spurious associations if not properly addressed [13].

Together, these challenges call for systems that combine advanced computational pipelines with nuanced biological understanding, enabling robust and generalizable GTA.

4 Method: GenoMAS

4.1 Overview

The complexity of gene expression analysis arises from both the high-dimensional, noisy nature of genomic data and the need for domain-specific decisions that critically affect downstream statistical outcomes. Conventional workflow-based approaches offer structured solutions but lack the flexibility needed to manage edge cases and persistent anomalies, resulting in suboptimal performance and efficiency in our evaluations (Section 5.3). In contrast, fully autonomous agents exhibit greater flexibility but fall short in delivering the precision essential for scientifically rigorous analyses, consistently failing to achieve acceptable performance on benchmark tasks (Appendix A). Existing methods thus remain insufficient, unable to reconcile the dual demands of *precise procedural control* and *robust error handling*—both of which are indispensable for trustworthy scientific automation.

GenoMAS introduces a multi-agent framework that unifies the precise control of traditional workflows with the adaptability of autonomous agents. At its core is a programming agent that interprets and executes user-defined guidelines while retaining the flexibility to address edge cases, retry failed operations, and incorporate domain-specific knowledge. This agent orchestrates a team of specialized collaborators, each endowed with distinct functional capabilities and coordinated through structured communication protocols and context-aware decision mechanisms.

4.2 Multi-Agent Architecture

Agent Roles and Responsibilities GenoMAS employs six specialized agents organized into three categories of different functions. (i) The **orchestration agent** is the *PI agent*, which coordinates the entire analysis workflow by dynamically assigning tasks based on analysis requirements and task dependencies. (ii) The **programming agents** perform the core computational tasks in multi-step workflows: two *Data Engineer agents* handle data preprocessing —the *GEO agent* focuses on Gene Expression Omnibus [22] datasets while the *TCGA agent* manages The Cancer Genome Atlas [111] data, each with specialized knowledge of their respective data formats and common preprocessing challenges. The *Statistician agent* conducts downstream statistical analyses, employing regression models for identifying trait-associated genes while accounting for confounding factors. (iii) The **advisory agents** support the programming agents through complementary expertise: the *Code Reviewer agent* validates generated code for functionality and instruction conformance, and provides suggestions for revision; while the *Domain Expert agent* provides biomedical insights for decisions requiring biological knowledge, such as clinical feature extraction and gene identifier mapping.

Diverse LLMs as Agents' Backbone Organizational science has established that cognitively diverse teams outperform homogeneous groups on complex tasks [43, 82]. This insight extends to multi-agent systems, where heterogeneous ensembles of models can achieve superior performance by leveraging complementary strengths [166]. Building on this principle, GenoMAS integrates a diverse set of state-of-the-art LLMs as backbones for its specialized agents. The programming agent is powered by Claude Sonnet 4 [5], selected for its strong agentic coding abilities. OpenAI o3 [80], known for its robust reasoning capabilities, serves dual roles—guiding the planning logic of programming agents and enabling the Code Reviewer to detect bugs and suggest targeted fixes. Gemini 2.5 Pro [27], one of the top-performing models on the GPQA [88] and HLE [61] benchmarks, serves as the backbone of the Domain Expert agent, providing broad and accurate scientific knowledge with particular strength in biology.

Task Orchestration The **PI agent** centrally orchestrates analysis workflows by adhering to the dependency structure intrinsic to gene expression analysis. For each GTA task, it identifies cohorts that have already been preprocessed and schedules preprocessing only for the missing data, thereby maximizing reuse across tasks with overlapping traits. Leveraging the independence of cohort-specific transformations, the agent parallelizes execution to substantially reduce runtime. Upon completion of preprocessing, the **Statistician agent** selects the most suitable cohorts for downstream inference. Throughout the pipeline, the PI agent issues tasks via structured messages, monitors completion signals, and enforces configurable timeouts to prevent unbounded execution. This coordinated orchestration ensures that agents operate methodically, efficiently, and within defined computational constraints.

Message-Driven Task Progression GenoMAS employs a typed message-passing protocol that governs task progression through structured request-response cycles. Messages carry four key elements: sender role, message type (e.g., `CODE_WRITING_REQUEST`, `CODE_REVIEW_RESPONSE`), content, and target roles. This design enables precise coordination among agents. For instance, upon completing a code segment, a programming agent sends a `CODE_REVIEW_REQUEST` targeting either the Code Reviewer or Domain Expert based on the action unit's requirements. The receiving agent processes the request and returns a typed response that determines the next action: approval advances to the next step, while rejection triggers revision. A centralized environment manages these interactions via a message queue, enforcing topological constraints to avoid circular dependencies. This communication framework enables task flows to emerge organically from agent interactions, while its structural rigor ensures coherent and goal-directed execution. Additional protocol details are provided in Appendix B.

4.3 Programming Agents

4.3.1 Guided Planning

Guidelines and Action Unit The Programming agents—**Data Engineers** and the **Statistician**—execute complex multi-step workflows using a guided planning framework that balances structure and flexibility. These agents operate under textual task guidelines that encode dependency structures, conditional logic, and termination criteria, effectively forming a directed acyclic graph (DAG). The guidelines serve as high-level execution blueprints, enabling agents to dynamically adjust their behavior based on the evolving context of each task.

Within this DAG, workflows are decomposed into *Action Units*: semantically coherent operations that correspond to discrete subtasks. For example, the GEO agent's workflow comprises Action Units for data loading, clinical feature extraction, gene annotation, and normalization. Each unit represents a self-contained sequence of operations that can be executed atomically, without requiring intermediate oversight. These Action Units are initially generated by the agent based on the guideline structure

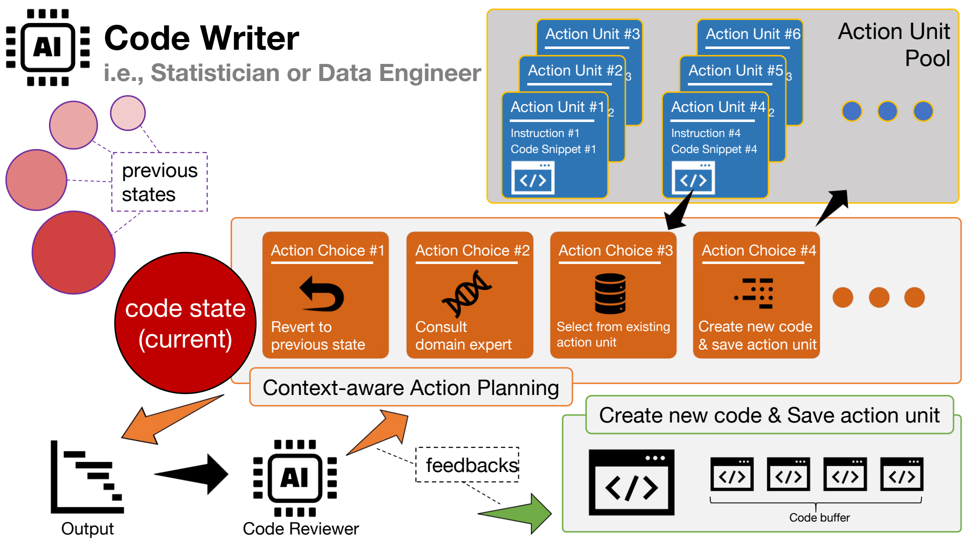


Figure 2: Planning, memory, and self-correction mechanisms of a single programming agent in our GenoMAS method.

and subsequently refined through manual curation to ensure correctness and completeness. See Appendix C for example guideline templates and Action Unit prompts for various task categories.

Guided Context-aware Planning At each step, programming agents analyze their task history and current state to select the next Action Unit. This planning process considers multiple factors: the success or failure of previous steps, the data characteristics discovered during execution, and the remaining task objectives. Agents can choose to proceed to the next logical Action Unit, revisit a previous step with modified parameters, skip optional steps when their preconditions are not met, or terminate the workflow when objectives are achieved.

The retraction mechanism allows agents to backtrack when they discover that an earlier decision led to downstream complications. Upon retraction, the agent reverts both its task context and execution state to a previous step, then continues with an alternative Action Unit. This capability proves essential for handling the complex interdependencies in gene expression analysis, where early preprocessing decisions can have cascading effects on data quality. Appendix D provides metaprompts and more details about our planning framework.

4.3.2 Domain Specific Code Generation

Iterative Code Generation and Debugging GenoMAS adopts a three-stage process—code writing, review, and revision—to generate robust analysis pipelines. For each Action Unit, programming agents receive the complete task context, including prior code executions, error traces, and historical attempts. This accumulated context allows agents to reason about existing data structures and avoid redundant mistakes. Upon code execution failure, the Code Reviewer evaluates the output, error messages, and adherence to task guidelines, issuing either an approval or a detailed rejection. Based on this feedback, programming agents refine and resubmit their code, iterating until approval is granted or a predefined debugging limit is reached.

To ensure independent evaluation, the code review process enforces context isolation. Code review Agent is provided with the current code attempt and overall task history but do not see feedback or decisions from previous review rounds at the same step. This design mitigates cascading biases and promotes objective assessments. After receiving a review response, the programming agent regains access to all prior attempts and feedback, allowing it to synthesize insights and revise the code accordingly. This separation supports both robust error detection and compliance with high-level task requirements.

Domain Expert Consultation for Biomedical Reasoning For Action Units requiring biomedical knowledge, programming agents can consult the Domain Expert agent in place of the Code Reviewer. The agent receives targeted context—such as metadata, processed summaries, and intermediate results, focusing on biological content rather than implementation details. The Domain Expert returns guidance in executable code form, enabling contextually grounded, biologically valid operations. This process is also iterative: execution failures are routed back to the same expert for debugging, facilitating consistent reasoning over multiple refinement rounds. Complex tasks may require several iterations to converge. Details and example prompts are provided in Appendix E.

Code Memory for Reuse Programming agents maintain a dynamic memory of validated code snippets indexed by Action Unit type. Upon successful review, a snippet is stored for potential reuse in similar contexts. This memory evolves with experience—agents can revise or replace stored code to reflect updated practices or domain shifts. The mechanism improves both efficiency and reliability by enabling the reuse of trusted patterns while preserving the flexibility to adapt to novel scenarios.

Specialized Tool Sets and Domain Knowledge Databases Each agent is equipped with a specialized set of Python tools aligned with its functional responsibilities. Data Engineers access utilities for dataset loading, DataFrame manipulation, and gene identifier mapping. The Statistician agent employs statistical models and visualization libraries. These tools were co-developed by a doctoral student and a computational biologist through iterative testing on real genomic datasets, ensuring both robustness and domain relevance. Agents use these tools flexibly—invoking them directly or adapting them to new contexts within an Action Unit—thus encapsulating best practices and simplifying code generation.

To ensure consistency and reduce external dependencies, GenoMAS integrates local, version controlled biological knowledge bases. A curated gene synonym database from NCBI Gene [12] supports accurate symbol normalization across naming conventions. Additionally, gene-trait association data from the Open Targets Platform [108] informs prioritization decisions during analysis. These resources are periodically updated to reflect current biomedical knowledge while ensuring reproducibility across experiments.

4.4 System Implementation and Optimizations

GenoMAS incorporates several system-level optimizations to support the practical demands of large-scale gene expression analysis. These enhancements address three critical dimensions: (i) *Efficiency*—the system leverages asynchronous LLM calls to enable concurrent agent operations and adopts memory-efficient data processing strategies, such as streaming pipelines and selective column loading, to prevent out-of-memory failures on large genomic datasets; (ii) *Robustness*—a task management framework tracks completed analyses and supports automatic workflow resumption following interruptions, while real-time resource monitoring and configurable timeouts safeguard against runaway processes; (iii) *Scalability*—mechanisms such as result caching and distributed task scheduling facilitate efficient execution across multiple GTA tasks.

Collectively, these design choices ensure that GenoMAS can process complex, real-world genomic data at scale while preserving the adaptability required for research-oriented exploration.

5 Experiments

We present the empirical evaluation of **GenoMAS** in comparison with baseline methods on the GenoTEX benchmark [66]. GenoTEX is currently the only comprehensive benchmark for gene expression analysis automation, encompassing 1,384 GTA problems across 913 datasets related to 132 human traits. It uniquely combines three core characteristics: (1) coverage of the complete analytical workflow from raw data to biological insights, (2) evaluation on actual genomic datasets with realistic complexities rather than simplified or structured data, and (3) expert-curated ground truth validated by professional bioinformaticians. This makes GenoTEX particularly suited for assessing the abilities of agentic methods to perform complex, multi-step analysis that requires scientific rigor.

5.1 Experimental Setup

Computation environment We conducted experiments on 6 RunPod GPU Cloud [91] instances, each provisioned with 16 vCPU cores and 94GB RAM. The data analysis code generated by agents consumed approximately 0.3GB CPU RAM during execution, remaining consistent across different LLM backbones. For LLM inference, we employed a hybrid deployment strategy: proprietary models (e.g., OpenAI o3, Claude Sonnet 4) were accessed via their official APIs, while open-source models (DeepSeek R1, Qwen 3 235B) were served through Novita AI's infrastructure [78], offering reduced latency compared to their respective official endpoints.

Metrics Our end-to-end evaluation assesses the ability of **GenoMAS** to identify significant genes by analyzing raw input data in GTA tasks. We adopt AUROC and F_1 scores, as detailed in Section 3.3, as primary indicators of analytical performance. To further validate the regression models produced by the system, we report binary trait prediction accuracy and F_1 scores on datasets selected by each method.

In addition to task-specific metrics, we track several runtime indicators. *Success rate* quantifies the proportion of GTA problems for which the generated analysis code successfully executes and saves results to the correct path. *Efficiency metrics* include input/output token counts, API-related costs, and average execution time per problem—offering a comprehensive view of system performance in real-world scenarios.

5.2 Comparison with Prior Art

We compare **GenoMAS** against the prior state-of-the-art system, GenoAgent [66], alongside a suite of baseline configurations. To assess the contribution of our heterogeneous LLM architecture, we evaluated homogeneous variants in which all agents share a single LLM backbone, testing multiple state-of-the-art models. As empirical reference points, we report the performance of random gene selection from [66], as well as human expert performance [66], computed as the average performance of two independent expert analyses when evaluated against a consensus ground truth. We also include a zero-shot baseline in which OpenAI o3 attempts to identify significant genes without access to data, probing the extent to which parametric knowledge alone suffices for this task.

Table 1 presents our end-to-end evaluation results. GenoMAS outperforms GenoAgent across nearly all metrics, yielding an absolute improvement of 16.85% in F_1 score, a 0.17 increase in AUROC, and a 44.7% reduction in API cost. These gains underscore the efficacy of our multi-agent design in addressing the complexity of automated gene expression analysis. Notably, the integration of heterogeneous LLMs proves essential: while Claude Sonnet 4 (Thinking) underpins our code generation, augmenting it with o3's reasoning abilities and Gemini 2.5 Pro's domain expertise yields an additional 7.5% improvement in F_1 and a 48.9% reduction in cost compared to homogeneous Claude-only configurations.

We observe that Biomni, despite its established success as a generalist biomedical agent through comprehensive tool integration, shows suboptimal performance in our evaluation (14.82% F_1). This performance gap likely stems from a fundamental difference in design philosophy: while Biomni leverages full agent autonomy to tackle diverse biomedical tasks with minimal user input, we target robust automation of more fully specified workflows with a higher demand for scientific rigor. As such, Biomni is not equipped with the context management mechanisms to dynamically plan each step based on task context and user guidelines while considering the complex inter-dependencies between steps. Furthermore, although Biomni provides local quality control through its self-critic mechanism, it is not designed to detect and recover from systematic errors at the global workflow level, where early methodological choices cascade through dozens of interdependent operations. These limitations underscore that reliable gene expression analysis requires the guided planning and structured orchestration that motivate our design of GenoMAS.

5.3 Ablation Study

Ablation on each component We conducted systematic ablation studies to quantify the contribution of each architectural component in **GenoMAS**: (i) Removing the planning mechanism forces execution along a fixed workflow defined by a static DAG, where agents retain access to task guidelines but cannot adapt execution order; (ii) Excluding the Domain Expert agent evaluates whether the Programming and Code Reviewer agents alone can handle biomedical reasoning; (iii) Restricting agents to a single review round assesses the impact of iterative refinement; and (iv) Omitting code review entirely measures baseline performance without quality control.

Empirical experiments highlight the significance of each component. The context-aware planning mechanism enables dynamic adaptation to edge cases and error recovery, yielding both higher accuracy

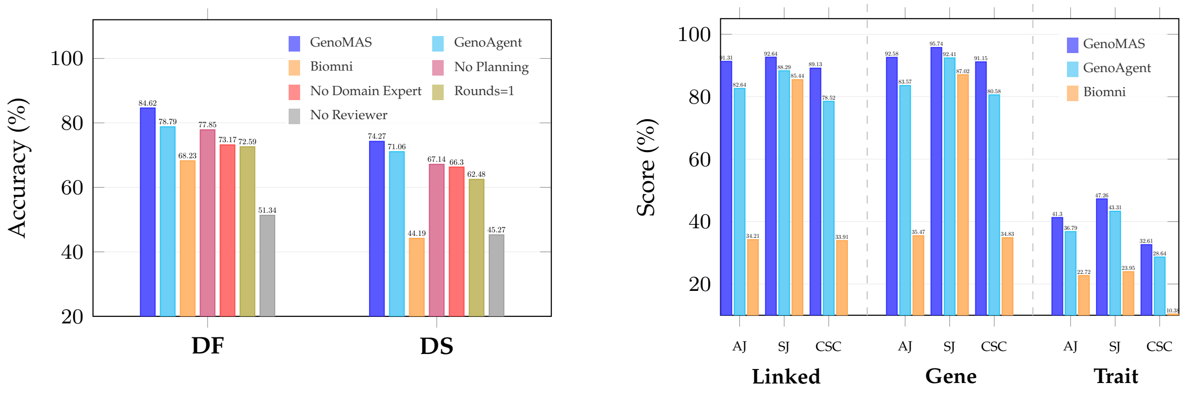
Table 1: **End-to-end performance of GenoMAS and prior art; performance of GenoMAS in homogeneous LLM setting with various LLMs; ablation results of GenoMAS; human expert performance, and simple data-free baselines.** We report performance on the GTA analysis problems, on the additional trait prediction task, and runtime performance with success rate and efficiency metrics. We use 0 as performance scores when code execution fails. "Tk.(K)", "Cost ($)", and "Time (s)" represent average input/output tokens (in thousands), API cost, and runtime per problem, respectively. "(T.)" indicates Extended Thinking mode.

Model	GTA Analysis					Trait Prediction		Runtime Performance			
	Prec.(%)	Rec.(%)	F_1(%)	AUROC	GSEA	Acc.(%)	F_1(%)	Succ.(%)	Tk.(K)	Cost($)	Time(s)
GenoMAS (Ours)	**61.75**	**66.67**	**60.48**	**0.81**	**0.95**	**86.25**	71.34	**98.78**	162.6/7.1	0.38	307.26
GenoAgent [66]	45.52	52.87	43.63	0.65	0.71	83.14	71.93	86.12	170.1/10.4	0.69	341.58
Biomni [48]	17.42	15.24	14.82	0.29	0.39	48.54	41.64	53.66	213.4/9.3	0.78	183.91
Claude Sonnet 4 (T.) [5]	53.36	55.88	52.98	0.74	0.93	85.77	**73.60**	96.34	177.4/13.2	0.73	282.85
Claude Sonnet 4 [5]	24.61	21.69	21.92	0.45	0.66	57.98	52.51	68.29	219.1/6.8	0.76	251.21
OpenAI o3 [80]	48.93	47.81	45.53	0.74	**0.95**	85.73	58.46	97.56	125.4/8.7	0.32	251.55
OpenAI o3-mini [81]	39.08	37.46	35.54	0.52	0.57	64.07	38.77	74.39	136.7/10.6	0.20	**127.37**
Gemini 2.5 Pro [27]	46.13	45.30	43.82	0.66	0.73	71.26	58.47	76.64	165.7/**3.3**	0.24	293.96
Gemini 2.5 Flash [26]	40.29	43.26	40.67	0.56	0.59	50.14	33.82	61.70	142.0/**3.3**	0.03	198.34
DeepSeek R1 [28]	13.42	13.58	13.45	0.13	0.14	11.23	8.95	13.94	107.9/28.7	0.15	851.23
Qwen 3 235B [87]	6.73	6.89	6.52	0.07	0.09	8.14	5.47	9.28	**78.0**/8.9	**0.02**	1084.51
No planning	51.61	57.01	51.27	0.74	0.87	73.32	59.90	89.59	204.1/9.3	0.48	359.79
No Domain Expert	50.06	52.72	47.57	0.69	0.81	64.75	50.14	85.24	182.6/8.0	0.43	322.54
Review Round=1	48.08	51.60	46.61	0.67	0.75	71.98	59.33	91.76	119.8/5.8	0.27	236.08
No reviewer	25.12	26.89	24.98	0.45	0.52	47.43	30.84	56.23	96.6/3.6	0.19	138.36
Human expert [66]	74.28	86.57	71.63	0.90	0.93	-	-	-	-	-	-
o3 zero-shot	15.84	4.28	5.13	0.56	0.41	-	-	-	0.1/0.8	0.01	12.18
Random [66]	0.72	0.90	0.75	0.49	0.04	-	-	-	-	-	0.02

Table 2: **Statistical analysis performance when using *expert-preprocessed data* as input.** We use the same evaluation metrics as in Table 1. Baselines include various representative agents and LLMs, and fixed scripts implementing two standard regression models for genomic data analysis. "BEC" in Row 2 represents batch effect correction.

Model	GTA Analysis					Trait Prediction		Runtime Performance			
	Prec.(%)	Rec.(%)	F_1(%)	AUROC	GSEA	Acc.(%)	F_1(%)	Succ.(%)	Tk.(K)	Cost($)	Time(s)
GenoMAS (Ours)	**97.14**	**92.87**	**95.26**	**0.97**	**0.99**	**88.37**	**77.21**	**100.00**	25.0/1.7	0.08	67.57
GenoMAS (No BEC)	70.18	75.60	69.64	0.85	0.92	86.48	61.29	**100.00**	24.2/1.6	0.07	65.34
GenoAgent [66]	94.81	92.27	93.83	0.96	0.97	87.23	72.67	99.71	28.5/1.9	0.09	71.60
Biomni [48]	90.56	88.56	89.23	0.92	0.94	85.05	71.87	96.52	26.1/1.4	0.10	37.83
Data Interpreter [45]	91.20	89.74	90.86	0.94	0.97	88.10	73.54	99.16	43.0/2.8	0.13	194.33
MetaGPT [44]	87.28	88.73	87.46	0.91	0.94	84.88	70.35	96.05	36.2/2.4	0.12	162.49
AutoGen [132]	86.04	87.09	86.38	0.93	0.95	82.81	72.24	96.60	41.7/2.6	0.71	139.48
CodeAct [121]	86.13	82.01	82.92	0.82	0.86	79.92	66.35	87.34	32.1/2.2	0.11	110.93
OpenAI o3 [80]	80.39	80.92	80.57	0.83	0.85	76.25	62.54	88.63	15.1/**1.5**	0.05	**39.64**
DeepSeek R1 [28]	73.42	73.18	73.65	0.74	0.76	74.31	57.23	78.56	**12.6**/4.9	**0.02**	136.22
Lasso [110]	31.88	12.99	14.03	0.53	0.12	76.48	62.31	99.39	-	-	9.19
LMM [42]	3.73	3.75	3.64	0.51	0.03	75.63	58.04	99.21	-	-	15.44

and greater efficiency by eliminating redundant steps and minimizing revision cycles. The collaborative design—particularly the inclusion of specialized Code Reviewer and Domain Expert agents—proves critical to maintaining scientific rigor. Allowing multiple review rounds further enhances reliability by catching subtle, downstream-impacting errors. In contrast, the zero-shot OpenAI o3 baseline achieves only 0.56 AUROC, underscoring that successful gene expression analysis requires structured data processing and domain reasoning, not merely parametric knowledge.



(a) Dataset filtering and selection

(b) Data preprocessing performance

Figure 3: **Individual task performance of GenoMAS and various baselines on dataset filtering, selection, and preprocessing tasks.** (a) Dataset filtering (DF) and selection (DS) accuracy of different methods, and different ablation settings of our GenoMAS method. (b) Data preprocessing performance across three data types (Linked, Gene, Trait) and three metrics (AJ: Attribute Jaccard, SJ: Sample Jaccard, CSC: Composite Similarity Correlation), with CSC as the primary metric.

Individual Task Performance To identify performance bottlenecks, we conducted a component-wise evaluation of the GenoMAS pipeline. Figure 3(a) presents results for the *dataset filtering and selection* stage. Agents exhibit reasonable effectiveness in this phase, likely due to the comparatively lower reasoning complexity involved in metadata-based relevance assessment. However, early-stage errors propagate through the pipeline, producing cascading effects that degrade overall performance.

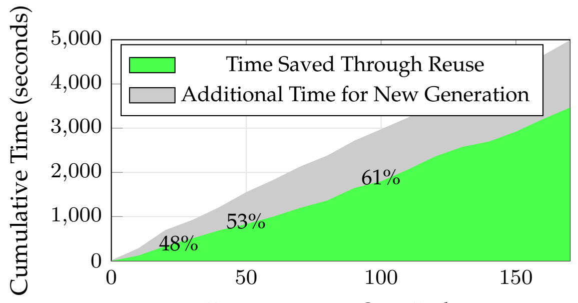
Data preprocessing (Figure 3(b)) eveal pronounced task-dependent variation, underscoring the inherent complexity of biological data analysis. **GenoMAS** achieves excellent performance on gene expression data (91.15% CSC), indicating its effectiveness in managing the technical intricacies of genomic data transformation. In contrast, clinical trait preprocessing yields a markedly lower CSC of 32.61%, a gap that reflects the heterogeneous nature of clinical data and the nuanced domain knowledge required for accurate extraction. This disparity identifies the preprocessing for clinical features as a main bottleneck where both technical sophistication and biomedical expertise are essential.

For *statistical analysis* (Table 2), we isolated this component by providing expert-preprocessed data, creating a controlled environment where methods primarily need to apply standard statistical libraries. In this setting, several agent-based approaches achieve competitive performance, yet important differences emerge. GenoMAS with batch effect correction achieves 95.26% F_1, substantially outperforming both traditional regression baselines (Lasso: 14.03%) and methods without systematic confound control. This result indicates that while basic statistical modeling is relatively straightforward for modern agents, the methodological sophistication to handle batch effects and covariate adjustment (capabilities built into our system through domain expertise) remains crucial for identifying genuinely significant biological signals.

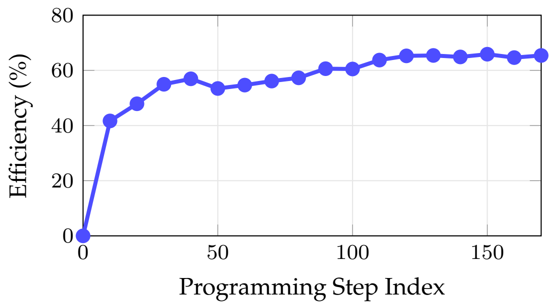
5.4 Memory and Code Reuse Efficiency

To understand how GenoMAS's memory mechanism contributes to its efficiency, we tracked code snippet reuse patterns during analysis of the first 50 cohort datasets. Figure 4 demonstrates the efficiency gain: the system's dynamic memory of validated code snippets saves 57.8 minutes during the tracked session, with an average of 20.3 seconds per reused programming step. The memory reuse rate stabilizes around 65% after initial learning, indicating that the system rapidly builds a reliable repertoire of reusable code patterns.

The memory mechanism's effectiveness arises because certain steps in gene expression analysis, such as loading GEO files, mapping gene symbols, and normalizing expression values, follow consistent patterns across datasets. By capturing these patterns in reusable code snippets, GenoMAS transforms redundant code generation into efficient lookups, allowing the system to allocate computational resources to novel, cohort-specific challenges.



(a) Cumulative time savings



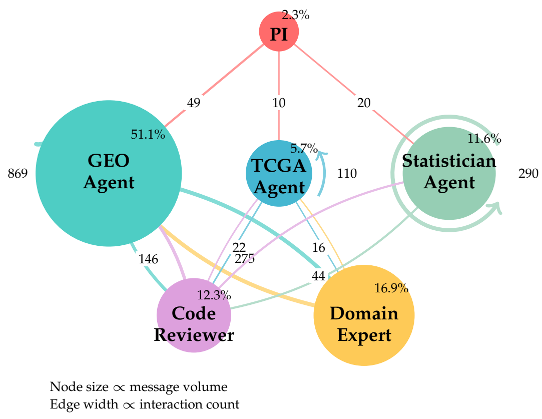
(b) Memory reuse rate evolution

Figure 4: **Memory reuse efficiency in GenoMAS.** (a) Cumulative time savings through memory reuse and (b) memory reuse rate evolution across programming steps. The system rapidly achieves high efficiency, stabilizing around 65% reuse rate after initial learning.

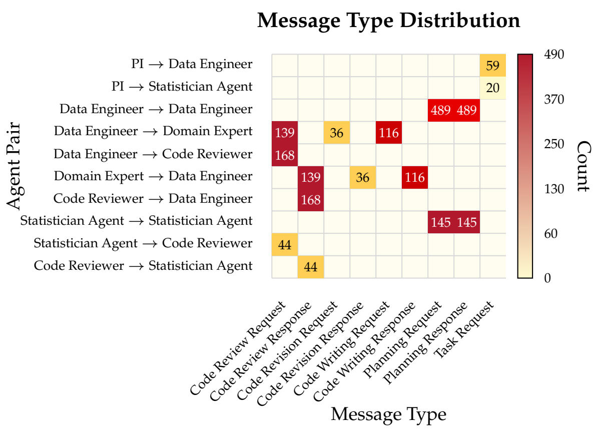
6 Qualitative Studies

Autonomous agent behaviors enhance workflow robustness Analysis of GenoMAS execution patterns reveals that agents autonomously adapt their behavior beyond prescribed instructions when encountering edge cases or persistent errors. Programming agents demonstrate context-aware problem-solving by proactively inserting diagnostic code (e.g., variable printing) to facilitate debugging, even without explicit reviewer guidance. In complex scenarios where an early decision creates downstream complications, agents can autonomously rewrite the entire code sequence starting from the problematic decision point, effectively correcting the execution trajectory without manual intervention. These emergent behaviors, documented in Appendix F, demonstrate how agent autonomy strengthens system robustness against the inherent variability of genomic data.

To further leverage the autonomy of our agents, we prompt them to generate structured post-task notes categorized by severity (info, warning, error), documenting analysis challenges, data anomalies, and potential issues. This self-reporting mechanism enables human experts to efficiently audit system decisions by focusing on high-priority error notes, creating a feedback loop for continuous system improvement. Representative examples are provided in Appendix G.



(a) Agent collaboration network



(b) Message type distribution heatmap

Figure 5: **Agent collaboration patterns in GenoMAS.** (a) Network topology showing agent communication structure with node size proportional to message volume. Edge thickness indicates interaction frequency. (b) Distribution of message types across agent pairs revealing asymmetric communication patterns, with programming agents predominantly sending validation requests while advisory agents respond with feedback. GEO and TCGA agents are merged into Data Engineer for brevity.

Multi-agent collaboration patterns reveal efficient task coordination Figure 5 visualizes GenoMAS's agent communication structure during a representative 20-problem analysis session. The Data Engineer (merged GEO and TCGA agents) dominates with 56.9% of interactions (1,956 messages), reflecting its central role in processing gene expression data, while the Statistician Agent accounts for 11.6% of interactions (398 messages) for analytical tasks. The PI agent's minimal 2.3% messaging efficiently orchestrates the workflow, with intensive bidirectional communication between programming and advisory agents enabling collaborative navigation of genomic complexities. These patterns highlight key insights for multi-agent systems: The Data Engineer's dominance emphasizes the benefits of role specialization, which centralizes intensive tasks while distributing expertise, mirroring organizational research on cognitive diversity [43] as manifested in our heterogeneous LLM architecture. The PI's low involvement demonstrates the system's high degree of autonomy (97.7% self-coordinated interactions), which contributes to a 44.7% API cost reduction over prior methods.

The heatmap reveals that Planning Requests/Responses (634 each) dominate, validating the role of our guided planning framework at every juncture of task execution. Code validation (351 requests) ranks second, with low error messages (36 revisions) indicating effective error prevention through multi-turn advisory mechanisms and guided planning, correlating with our 98.78% success rate.

Overall, these metrics reveal design principles for computational biology agents: balancing centralized execution with distributed expertise minimizes overhead and boosts adaptability, ultimately enabling scalable analysis that outperforms prior art by 16.85% in F_1 score.

7 Conclusion

We presented **GenoMAS**, a multi-agent system that advances automated gene expression analysis through three key methodological contributions: guided planning that balances structured workflows with autonomous adaptation, heterogeneous LLM architecture that leverages complementary strengths across coding, reasoning, and domain expertise, and dynamic memory mechanisms that enable efficient code reuse across analyses. These design principles address the fundamental tension between precise procedural control and adaptive error handling, a challenge that has limited prior approaches to this domain. Through extensive evaluation on the GenoTEX benchmark, GenoMAS demonstrates substantial empirical improvements, achieving a 60.48% F_1 score in gene-trait association analysis (16.85% absolute gain over prior work) while reducing computational costs by 44.7%. Our qualitative analysis confirms that GenoMAS identifies biologically meaningful associations supported by existing literature.

As genomic data continues to expand exponentially, we envision GenoMAS democratizing sophisticated bioinformatics analyses, enabling researchers across disciplines to extract insights from complex molecular data while maintaining the precision essential for scientific discovery. Beyond genomics, the principles underlying our approach (guided planning, cognitive diversity, and domain-informed programming) may inspire similar frameworks for other complex scientific domains. Future work will explore multi-modal biological data integration and more sophisticated planning algorithms, while continuing to prioritize the balance between automation capabilities and scientific trustworthiness that defines responsible AI for research.

Acknowledgements

This research was supported by the National AI Research Resource (NAIRR) under grant number 240283.

GenoMAS: A Multi-Agent Framework for Scientific Discovery via Code-Driven Gene Expression Analysis

Supplementary Material

The supplementary material is organized as follows:

- Appendix A presents test cases showing fundamental limitations of existing autonomous agent methods on gene expression data analysis.
- Appendix B describes the communication protocols and messaging mechanisms in GenoMAS.
- Appendix C presents the task-specific guidelines and action unit prompts for GEO and TCGA preprocessing.
- Appendix D details the guided planning framework and metaprompts for programming agents.
- Appendix E presents the code generation, review, and domain consultation mechanisms with their metaprompts.
- Appendix F presents some examples of agent autonomy in GenoMAS.
- Appendix G presents some examples of notes written by agents.

A Fundamental Limitations of Existing Autonomous Agent Methods

Recent advances in LLM-based agents have introduced numerous methods and products promising general-purpose automation with full autonomy. These systems handle task decomposition, planning, and multi-agent collaboration with minimal user input. To evaluate their applicability to gene expression analysis, we systematically tested six representative agents: three open-source systems (Biomni [48], AutoGen [132], MetaGPT [44]) and three proprietary systems (Claude Code [4], Cursor [6], Manus [98]). Our experiments reveal that all these methods face fundamental limitations when applied to gene expression data analysis due to insufficient mechanisms for precise procedural control.

Among tested agents, Biomni incorporates 150 specialized biomedical tools, 105 software packages, and 59 databases with carefully designed tool retrieval mechanisms. This comprehensive biomedical specialization demonstrates the strongest domain adaptation among tested systems, yielding an F_1 score of 14.82% in our experiments (Table 1 in the main paper), substantially higher than other agents but still far below GenoMAS (60.48%) and the previous best method GenoAgent (43.63%). The remaining five agents, designed for open-domain or general programming tasks, consistently produce invalid analyses across test scenarios. Their failure rates prevent meaningful inclusion in quantitative benchmarks for end-to-end analysis.

Our analysis identifies two critical limitations in existing autonomous agents:

Insufficient guidance interpretation: General prompts prove inadequate for guiding agents through the complex analytical requirements of gene expression data. Agents frequently make consequential errors early in the workflow that propagate through subsequent steps, resulting in scientifically invalid outputs. This demonstrates the necessity of incorporating user-specified, domain-trusted guidelines into agent architectures.

Absence of procedural control mechanisms: Even when provided with detailed guidelines and Action Unit specifications, current agents lack mechanisms to ensure adherence to prescribed workflows. They cannot perform context-aware planning to identify deviations from expected procedures or recover from errors at the workflow level. This absence of precise procedural control renders them unsuitable for automation of gene expression analysis, where methodological rigor directly impacts scientific validity.

The following sections present concrete examples demonstrating these limitations in preprocessing GEO datasets. We focus on three cohorts for the trait 'Hypertension': GSE77627, GSE117261, and GSE256539. These cohorts contain complete information necessary for analysis and were successfully preprocessed by both human experts in the benchmark [66] and GenoMAS. For each agent, we provide results from two prompt conditions: (1) general instructions and (2) detailed instructions augmented with our guidelines and Action Unit specifications (provided in Appendix C).

A.1 Prompt Conditions

A.1.1 General Prompt

We use the following general prompt, with minor modifications to adapt to the agent-specific I/O interface:

```
Please help me preprocess the cohort dataset.
I need to get a complete, linked dataset where each row represents a sample and each column
     represents a feature.
The row index should be the sample ID. The columns should include and only include the
    following:
- The trait represented as a binary or continuous variable, whichever is more suitable. If
    it is binary, use 1 to represent positive and 0 as control. Use the trait name as the
    column name.
- The sample's age (if available), a float or integer representing the age in years. Use "
    Age" as the column name.
- The sample's gender (if available), a binary variable with 0 for female and 1 for male.
    Use "Gender" as the column name.
- The expression values of genes, with gene symbols as column names.
```

A.1.2 Detailed Prompt

The general prompt augmented with our task-specific guidelines, followed by the reminder:

```
Below are more detailed descriptions about some of the steps in the guidelines. You are
    free to selectively execute all or some of them based on task context, and decide the
    order that is best for completing the task in accordance with the above guidelines.
```

This reminder is then followed by the Action Unit instructions, emphasizing agents' flexibility to plan and selectively execute based on context.

A.2 Biomni Examples

Despite incorporating specialized biomedical tools, Biomni demonstrates fundamental limitations in understanding disease biology and correctly interpreting clinical phenotypes. The following example from GSE77627 illustrates how these limitations manifest in both prompt conditions.

A.2.1 GSE77627: Misinterpretation of Portal Hypertension Phenotypes

This dataset studies portal hypertension, a condition characterized by elevated blood pressure in the portal venous system. The study includes three liver groups: HNL (Histologically Normal Livers), INCPH (Idiopathic Non-Cirrhotic Portal Hypertension), and LC (Liver Cirrhosis). Both INCPH and LC represent different etiologies of portal hypertension, while HNL serves as the healthy control group.

Under the general prompt condition, Biomni implements the following erroneous trait conversion:

```
def convert_trait(value):
    """Convert trait values to binary (INCPH=1, others=0)"""
    if value is None:
        return None
    # Extract value after colon
    if ':' in str(value):
        val = str(value).split(':')[1].strip()
    else:
        val = str(value).strip()

    # Convert INCPH to 1 (positive), others to 0 (control)
    if val == 'INCPH':
        return 1
    elif val in ['HNL', 'LC']:  # Critical error: LC is portal hypertension, not control
        <--
        return 0
    else:
```