

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

Haoyang Liu¹, Yijiang Li², and Haohan Wang¹

¹University of Illinois at Urbana-Champaign

²University of California, San Diego

{h157, 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

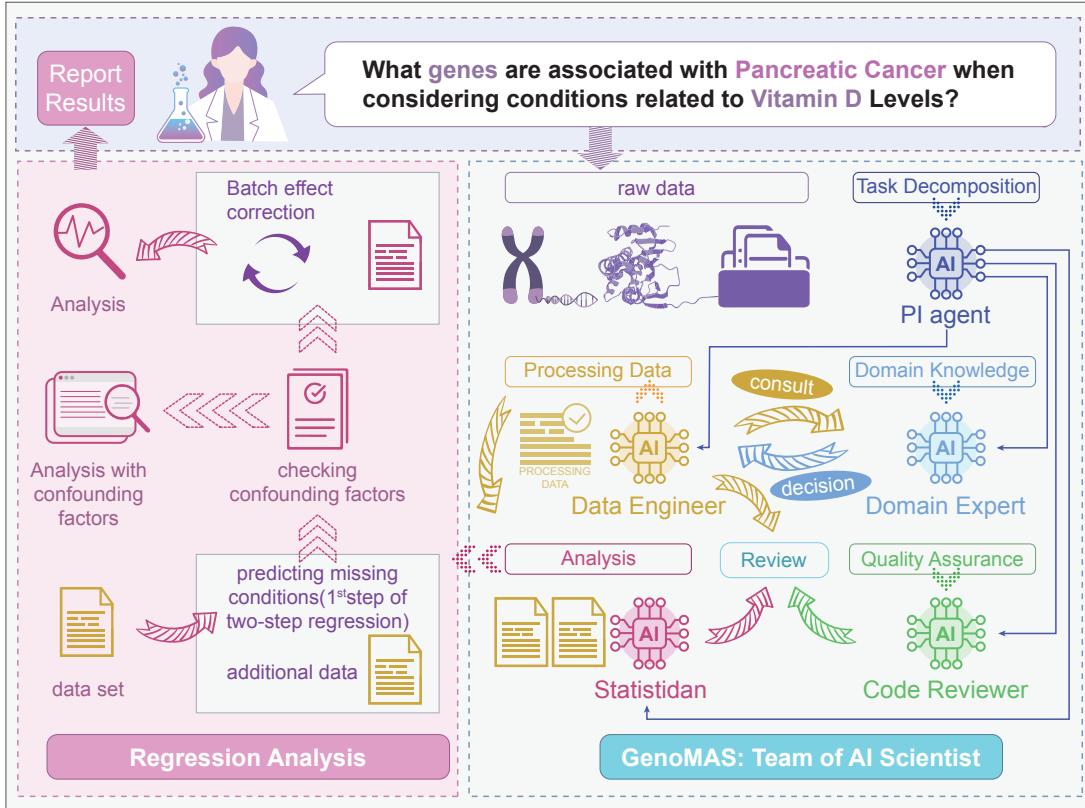


Figure 1: Multi-agent collaboration in our GenoMAS method.

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** (Genomic data analysis through LLM-based Multi-Agent System), 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 $G^* \subseteq \{1, \dots, p\}$ 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 platforms—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) = $AJ \times SJ \times \text{Corr}_{\text{avg}}$, 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 include 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 AUROC [40], appropriate for the highly imbalanced setting (often <2% significant genes), and is complemented by Precision, Recall, F₁, 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

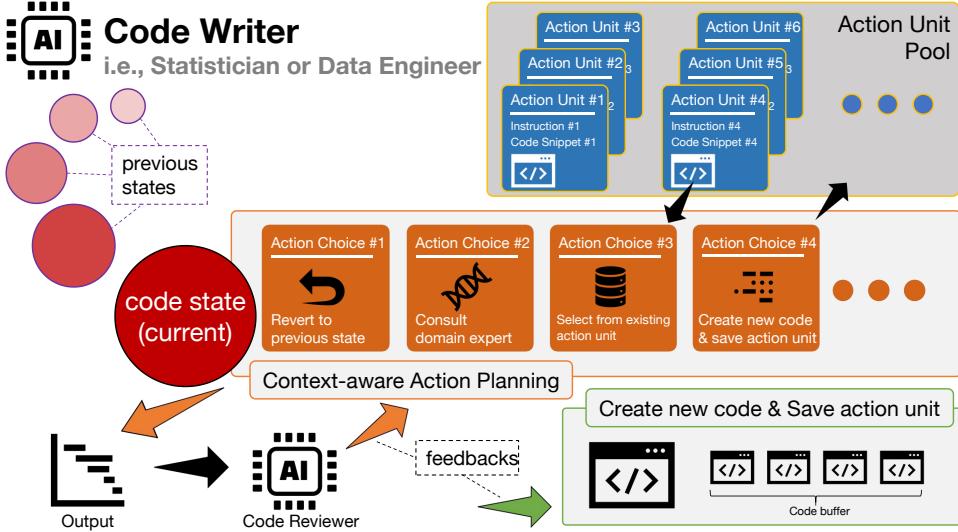


Figure 2: Planning, memory, and self-correction mechanisms of a single programming agent in our GenoMAS method.

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.

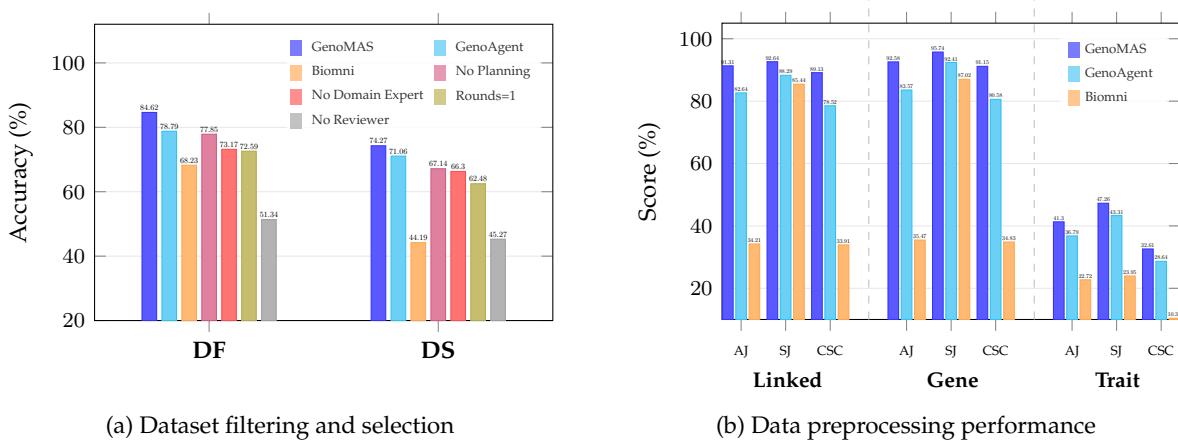


Figure 3: Individual task performance of GenoMAS and various baselines on dataset filtering, selection, and preprocessing tasks. (a) Dataset filtering (DF) and selection (DS) accuracy of different methods, and different ablation settings of our GenoMAS method. (b) Data preprocessing performance across 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)) reveal 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% F₁, 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.

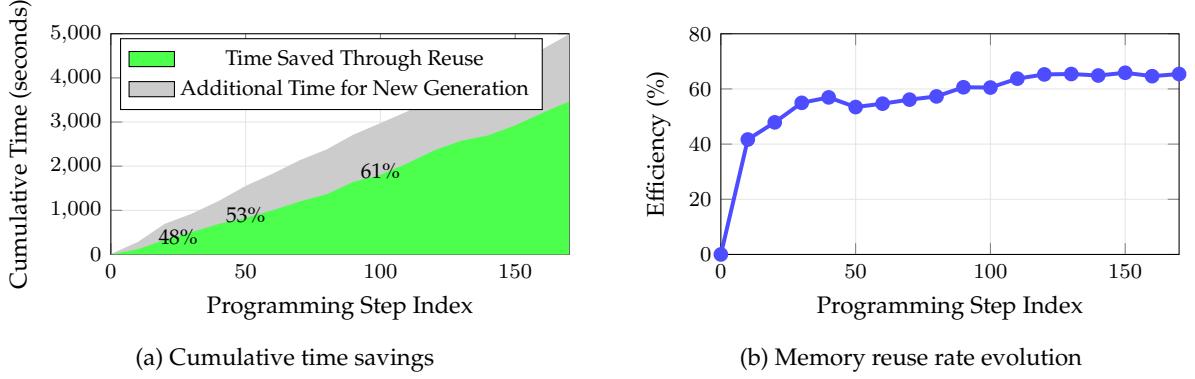


Figure 4: **Memory reuse efficiency in GenoMAS.** (a) Cumulative time savings through memory reuse and (b) memory reuse rate evolution across programming steps. The system rapidly achieves high efficiency, stabilizing around 65% reuse rate after initial learning.

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.

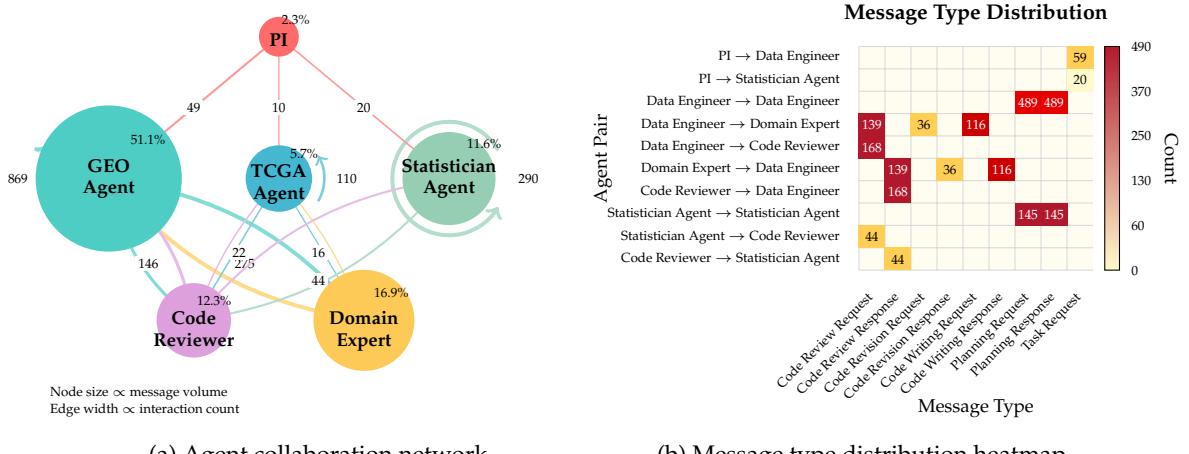


Figure 5: **Agent collaboration patterns in GenoMAS.** (a) Network topology showing agent communication structure with node size proportional to message volume. Edge thickness indicates interaction frequency. (b) Distribution of message types across agent pairs revealing asymmetric communication patterns, 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 F₁ 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% F₁ 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.

References

- [1] H. Abusamra. A comparative study of feature selection and classification methods for gene expression data of glioma. *Procedia Computer Science*, 23:5–14, 2013.
- [2] T. Aittokallio. Dealing with missing values in large-scale studies: microarray data imputation and beyond. *Briefings in bioinformatics*, 11(2):253–264, 2010.
- [3] C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle. Deep learning for computational biology. *Molecular systems biology*, 12(7):878, 2016.
- [4] Anthropic. Claude code: Agentic coding tool, 2024. URL <https://www.anthropic.com/clause/code>. Command line tool for agentic coding.
- [5] Anthropic. Introducing claude 4: Our most intelligent model, 2024. URL <https://www.anthropic.com/clause>. Accessed: 2025-01-22.
- [6] Anysphere. Cursor: The ai code editor, 2024. URL <https://cursor.com>. AI-powered code editor.
- [7] K. Baba, C. Liu, S. Kurita, and A. Sannai. Prover agent: An agent-based framework for formal mathematical proofs. *arXiv preprint arXiv:2506.19923*, 2025.
- [8] J. Baek, S. K. Jauhar, S. Cucerzan, and S. J. Hwang. Researchagent: Iterative research idea generation over scientific literature with large language models. *arXiv preprint arXiv:2404.07738*, 2024.
- [9] J. L. Ballard, Z. Wang, W. Li, L. Shen, and Q. Long. Deep learning-based approaches for multi-omics data integration and analysis. *BioData Mining*, 17(1):38, 2024. doi: 10.1186/s13040-024-00391-z.
- [10] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, L. Gianinazzi, J. Gajda, T. Lehmann, M. Podstawski, H. Niewiadomski, P. Nyczyk, and T. Hoefer. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv: 2308.09687*, 2023.
- [11] A. M. Bran, S. Cox, O. Schilter, C. Baldassari, A. D. White, and P. Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv: 2304.05376*, 2023.
- [12] G. R. Brown, V. Hem, K. S. Katz, M. Ovetsky, C. Wallin, O. Ermolaeva, I. Tolstoy, T. Tatusova, K. D. Pruitt, and D. R. Maglott. Gene: a gene-centered information resource at NCBI. *Nucleic Acids Research*, 43(D1):D36–D42, 2015. doi: 10.1093/nar/gku1055. URL <https://doi.org/10.1093/nar/gku1055>.
- [13] O. Bruning, W. Rodenburg, P. F. Wackers, C. Van Oostrom, M. J. Jonker, R. J. Dekker, H. Rauwerda, W. A. Ensink, A. De Vries, and T. M. Breit. Confounding factors in the transcriptome analysis of an in-vivo exposure experiment. *PLoS One*, 11(1):e0145252, 2016.
- [14] S. A. Byron, K. R. Van Keuren-Jensen, D. M. Engelthaler, J. D. Carpten, and D. W. Craig. Translating rna sequencing into clinical diagnostics: opportunities and challenges. *Nature Reviews Genetics*, 17(5):257–271, 2016. doi: 10.1038/nrg.2016.10.
- [15] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran, M. Zaharia, J. E. Gonzalez, and I. Stoica. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.
- [16] V. Çetin and O. YILDIZ. A comprehensive review on data preprocessing techniques in data analysis. *Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi*, 28(2):299–312, 2022.
- [17] I. S. Chan and G. S. Ginsburg. Personalized medicine: progress and promise. *Annual review of genomics and human genetics*, 12:217–244, 2011.
- [18] K. Chen, Y. Zhou, X. Zhang, and H. Wang. Prompt stability matters: Evaluating and optimizing auto-generated prompt in general-purpose systems. *arXiv preprint arXiv:2505.13546*, 2025.
- [19] X. Chen, M. Lin, N. Schärli, and D. Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv: 2304.05128*, 2023.
- [20] X. Chen, T. Wang, T. Liu, Z. Guo, X. Li, M. Qu, and T. Zhao. A survey on hypothesis generation for scientific discovery in the era of large language models. *arXiv preprint arXiv:2504.05496*, 2025.

- [21] Z. Chen, L. Cao, S. Madden, T. Kraska, Z. Shang, J. Fan, N. Tang, Z. Gu, C. Liu, and M. Cafarella. Seed: Domain-specific data curation with large language models. *arXiv preprint arXiv:2310.00749*, 2023.
- [22] E. Clough and T. Barrett. The gene expression omnibus database. *Methods in Molecular Biology*, 1418:93–110, 2016. doi: 10.1007/978-1-4939-3578-9_5.
- [23] A. Conesa, P. Madrigal, S. Tarazona, D. Gomez-Cabrero, A. Cervera, A. McPherson, M. W. Szczęśniak, D. J. Gaffney, L. L. Elo, X. Zhang, and A. Mortazavi. A survey of best practices for rna-seq data analysis. *Genome Biology*, 17:13, 2016. doi: 10.1186/s13059-016-0881-8.
- [24] J. P. Cook, A. Mahajan, and A. P. Morris. Guidance for the utility of linear models in meta-analysis of genetic association studies of binary phenotypes. *European Journal of Human Genetics*, 25(2):240–245, 2017.
- [25] T. Dai, S. Vijayakrishnan, F. T. Szczypinski, J.-F. Ayme, E. Simaei, T. Fellowes, R. Clowes, L. Kotopanov, C. E. Shields, Z. Zhou, J. W. Ward, and A. I. Cooper. Autonomous mobile robots for exploratory synthetic chemistry. *Nature*, pages 1–8, Nov. 2024. ISSN 1476-4687. doi: 10.1038/s41586-024-08173-7.
- [26] G. DeepMind. Gemini 2.5 flash, 2025. URL <https://deepmind.google/models/gemini/flash/>. Fast performance thinking model for everyday tasks.
- [27] G. DeepMind. Gemini 2.5 pro, 2025. URL <https://deepmind.google/models/gemini/>. Advanced thinking model with Deep Think mode for complex reasoning tasks.
- [28] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. Chen, R. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, and S. Li. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [29] Y. Dong, X. Jiang, Z. Jin, and G. Li. Self-collaboration code generation via chatgpt. *arXiv preprint arXiv: 2304.07590*, 2023.
- [30] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv: 2305.14325*, 2023.
- [31] R. Edgar, M. Domrachev, and A. E. Lash. Ncbi geo: archive for gene expression and epigenomics data sets: 23-year update. *Nucleic Acids Research*, 52(D1):D138–D144, 2024.
- [32] A. Espín-Pérez, C. Portier, M. Chadeau-Hyam, K. van Veldhoven, J. C. Kleinjans, and T. M. de Kok. Comparison of statistical methods and the use of quality control samples for batch effect correction in human transcriptome data. *PloS one*, 13(8):e0202947, 2018.
- [33] G. Feng, B. Zhang, Y. Gu, H. Ye, D. He, and L. Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective. *NEURIPS*, 2023.
- [34] J. A. Gagnon-Bartsch and T. P. Speed. Using control genes to correct for unwanted variation in microarray data. *Biostatistics*, 13(3):539–552, 2012. doi: 10.1093/biostatistics/kxr034.
- [35] D. Ghosh and A. M. Chinnaiyan. Classification and selection of biomarkers in genomic data using lasso. *Journal of Biomedicine and Biotechnology*, 2005(2):147, 2005.
- [36] G. S. Ginsburg and K. A. Phillips. Precision medicine: from science to value. *Health Affairs*, 37(5): 694–701, 2018. doi: 10.1377/hlthaff.2017.1624.
- [37] Z. Gou, Z. Shao, Y. Gong, Y. Shen, Y. Yang, N. Duan, and W. Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.
- [38] T. Guo, K. Guo, B. Nan, Z. Liang, Z. Guo, N. V. Chawla, O. Wiest, and X. Zhang. What can large language models do in chemistry? a comprehensive benchmark on eight tasks. *arXiv preprint*

arXiv:2305.18365, 2023.

- [39] M. A. Hamburg and F. S. Collins. The path to personalized medicine. *New England Journal of Medicine*, 363(4):301–304, 2010.
- [40] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.
- [41] S. Hao, Y. Gu, H. Ma, J. J. Hong, Z. Wang, D. Wang, and Z. Hu. Reasoning with language model is planning with world model. *Conference on Empirical Methods in Natural Language Processing*, 2023. doi: 10.48550/arXiv.2305.14992.
- [42] C. R. Henderson. Estimation of genetic parameters. *Biometrics*, 6(2):186–190, 1950. doi: 10.2307/3001414.
- [43] L. Hong and S. E. Page. Groups of diverse problem solvers can outperform groups of high-ability problem solvers. *Proceedings of the National Academy of Sciences*, 101(46):16385–16389, 2004.
- [44] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv: 2308.00352*, 2023.
- [45] S. Hong, Y. Lin, B. Liu, B. Liu, B. Wu, C. Zhang, C. Wei, D. Li, J. Chen, J. Zhang, J. Wang, L. Zhang, L. Zhang, M. Yang, M. Zhuge, T. Guo, T. Zhou, W. Tao, X. Tang, X. Lu, X. Zheng, X. Liang, Y. Fei, Y. Cheng, Z. Gou, Z. Xu, and C. Wu. Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679*, 2024.
- [46] S. Hu, C. Lu, and J. Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.
- [47] J. Huang, X. Chen, S. Mishra, H. S. Zheng, A. W. Yu, X. Song, and D. Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.
- [48] K. Huang, S. Zhang, H. Wang, A. Bhattacharjee, Y. Lu, M. Wen, J. Yang, and J. Ye. Biomni: A general-purpose biomedical ai agent. *bioRxiv preprint bioRxiv:2025.05.30.656746*, 2025.
- [49] Y. Huang, J. Shi, Y. Li, C. Fan, S. Wu, Q. Zhang, Y. Liu, P. Zhou, Y. Wan, N. Z. Gong, and L. Sun. Metatool benchmark for large language models: Deciding whether to use tools and which to use, 2024. URL <https://arxiv.org/abs/2310.03128>.
- [50] S. Jia, T. Huo, and Y. Zeng. Llmatdesign: Autonomous materials discovery with large language models. *arXiv preprint arXiv:2406.13163*, 2024.
- [51] W. E. Johnson, C. Li, and A. Rabinovic. Adjusting batch effects in microarray expression data using empirical bayes methods. *Biostatistics*, 8(1):118–127, 2007. doi: 10.1093/biostatistics/kxj037.
- [52] I. M. Johnstone. On the distribution of the largest eigenvalue in principal components analysis. *The Annals of statistics*, 29(2):295–327, 2001.
- [53] H. B. Kang, N. Soliman, M. Latzke, J. C. Chang, and J. Bragg. Comlittee: Literature discovery with personal elected author committees. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–20, 2023.
- [54] M. M. R. Khondoker. *Statistical methods for pre-processing microarray gene expression data*. PhD thesis, University of Edinburgh, 2006.
- [55] B. Kyalwazi, C. Yau, M. J. Campbell, T. F. Yoshimatsu, A. J. Chien, A. M. Wallace, A. Forero-Torres, L. Pusztai, E. D. Ellis, K. S. Albain, et al. Race, gene expression signatures, and clinical outcomes of patients with high-risk early breast cancer. *JAMA Network Open*, 6(12):e2349646–e2349646, 2023.
- [56] E. Latif, R. Parasuraman, and X. Zhai. Physicsassistant: An llm-powered interactive learning robot for physics lab investigations. In *2024 33rd IEEE International Conference on Robot and Human Interactive Communication (ROMAN)*, pages 864–871. IEEE, 2024.
- [57] J. T. Leek, R. B. Scharpf, H. C. Bravo, D. Simcha, B. Langmead, W. E. Johnson, D. Geman, K. Baggerly, and R. A. Irizarry. Tackling the widespread and critical impact of batch effects in high-throughput data. *Nature Reviews Genetics*, 11(10):733–739, 2010.
- [58] B. Li and C. N. Dewey. Rsem: accurate transcript quantification from rna-seq data with or without

- a reference genome. *BMC Bioinformatics*, 12:323, 2011. doi: 10.1186/1471-2105-12-323.
- [59] H. Li, Y. Q. Chong, S. Stepputtis, J. Campbell, D. Hughes, M. Lewis, and K. Sycara. Theory of mind for multi-agent collaboration via large language models. *arXiv preprint arXiv:2310.10701*, 2023.
- [60] L. Li, W. Xu, J. Guo, R. Zhao, X. Li, Y. Yuan, B. Zhang, Y. Jiang, Y. Xin, R. Dang, et al. Chain of ideas: Revolutionizing research via novel idea development with llm agents. *arXiv preprint arXiv:2410.13185*, 2024.
- [61] Y. Li, Y. Zhang, and X. Chen. Hle-bench: A holistic evaluation benchmark for large language models in higher-level reasoning. *arXiv preprint arXiv:2406.10833*, 2024.
- [62] A. W.-C. Liew, N.-F. Law, and H. Yan. Missing value imputation for gene expression data: computational techniques to recover missing data from available information. *Briefings in bioinformatics*, 12(5):498–513, 2011.
- [63] C. Lippert, J. Listgarten, Y. Liu, C. M. Kadie, R. I. Davidson, and D. Heckerman. Fast linear mixed models for genome-wide association studies. *Nature methods*, 8(10):833–835, 2011.
- [64] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone. Llm+p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv: 2304.11477*, 2023.
- [65] B. Liu, X. Li, J. Zhang, J. Wang, T. He, S. Hong, H. Liu, S. Zhang, K. Song, K. Zhu, Y. Cheng, S. Wang, X. Wang, Y. Luo, H. Jin, P. Zhang, O. Liu, J. Chen, H. Zhang, Z. Yu, H. Shi, B. Li, D. Wu, F. Teng, X. Jia, J. Xu, J. Xiang, Y. Lin, T. Liu, T. Liu, Y. Su, H. Sun, G. Berseth, J. Nie, I. Foster, L. Ward, Q. Wu, Y. Gu, M. Zhuge, X. Tang, H. Wang, J. You, C. Wang, J. Pei, Q. Yang, X. Qi, and C. Wu. Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems, 2025. URL <https://arxiv.org/abs/2504.01990>.
- [66] H. Liu, S. Chen, Y. Zhang, and H. Wang. Genotex: A benchmark for automated gene expression data analysis in alignment with bioinformaticians, 2025. URL <https://arxiv.org/abs/2406.15341>.
- [67] S. Liu, Y. Lu, S. Chen, X. Hu, J. Zhao, Y. Lu, and Y. Zhao. Drugagent: Automating ai-aided drug discovery programming through llm multi-agent collaboration. *arXiv preprint arXiv:2411.15692*, 2024.
- [68] Z. Liu, Y. Zhang, P. Li, Y. Liu, and D. Yang. Dynamic LLM-agent network: An LLM-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*, 2023.
- [69] Z. Liu, Y. Huang, S. Raman, A. Anandamurthy, V. Makeeva, V. Subbotin, D. Grushevskaya, K. Raman, E. Kalabusheva, J. Bagaitkar, T. Cui, B. Ren, M. Shvedova, J. Attie, C. Weng, P. Dolzhenko, M. J. Martinez, and K. Zhang. Transcriptomics and epigenetic data integration learning module on google cloud. *Briefings in Bioinformatics*, 25(Supplement_1):bbae352, 2024. doi: 10.1093/bib/bbae352.
- [70] M. I. Love, W. Huber, and S. Anders. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, 15(12):550, 2014.
- [71] C. Lu, C. Lu, R. T. Lange, J. Foerster, J. Clune, and D. Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.
- [72] H. Ma, T. Hu, Z. Pu, L. Boyin, X. Ai, Y. Liang, and M. Chen. Coevolving with the other you: Fine-tuning llm with sequential cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 37:15497–15525, 2024.
- [73] P. Ma, T.-H. Wang, M. Guo, Z. Sun, J. B. Tenenbaum, D. Rus, C. Gan, and W. Matusik. Llm and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. *arXiv preprint arXiv:2405.09783*, 2024.
- [74] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhummoye, Y. Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.
- [75] A. Madani, B. Krause, E. R. Greene, S. Subramanian, B. P. Mohr, J. M. Holton, J. L. Olmos Jr,

- C. Xiong, Z. Z. Sun, R. Socher, et al. Large language models generate functional protein sequences across diverse families. *Nature Biotechnology*, pages 1–8, 2023.
- [76] J. D. Martin-Rufino, A. Caulier, L. E. Torres, A. Babu, S. Li, S. H. Jung, D. B. Keskin, X. Wang, S. Saori, P. Giuliana, M. Gu, A. A. Thompson, V. G. Sankaran, and E. S. Lander. Transcription factor networks disproportionately enrich for heritability of blood cell phenotypes. *Science*, 388 (6666):52–59, 2025. doi: 10.1126/science.ads7951.
- [77] X. Ning, Z. Lin, Z. Zhou, H. Yang, and Y. Wang. Skeleton-of-thought: Large language models can do parallel decoding. *arXiv preprint arXiv:2307.15337*, 2023.
- [78] Novita AI. Novita ai: Deploy ai models effortlessly with our simple api. <https://novitaai.com>, 2025. Accessed: 2025-02-17.
- [79] OpenAI. Gpt-4 technical report. PREPRINT, 2023.
- [80] OpenAI. Openai o3, 2025. URL <https://openai.com/o3>. Accessed: 2025-01-22.
- [81] OpenAI. Openai o3-mini, 2025. URL <https://openai.com/index/openai-o3-mini/>. Latest Large language model of OpenAI.
- [82] S. E. Page. The difference: How the power of diversity creates better groups, firms, schools, and societies. *Princeton University Press*, 2007.
- [83] J. Park, J. C. O'Brien, C. J. Cai, M. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. *ACM Symposium on User Interface Software and Technology*, 2023. doi: 10.1145/3586183.3606763.
- [84] C. Qian, X. Cong, W. Liu, C. Yang, W. Chen, Y. Su, Y. Dang, J. Li, J. Xu, D. Li, Z. Liu, and M. Sun. Communicative agents for software development. *arXiv preprint arXiv: 2307.07924*, 2023.
- [85] C. Qian, E. C. Acikgoz, Q. He, H. Wang, X. Chen, D. Hakkani-Tür, G. Tur, and H. Ji. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025.
- [86] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, S. Zhao, L. Hong, R. Tian, R. Xie, J. Zhou, M. Gerstein, D. Li, Z. Liu, and M. Sun. Toollm: Facilitating large language models to master 16000+ real-world apis, 2023. URL <https://arxiv.org/abs/2307.16789>.
- [87] Qwen Team. Qwen3-235b-a22b, 2025. URL <https://github.com/QwenLM/Qwen3>. A 235B parameter Mixture-of-Experts model with 22B activated parameters by Alibaba Cloud.
- [88] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.
- [89] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.
- [90] D. Rosati, M. Palmieri, G. Brunelli, A. Morrione, F. Iannelli, E. Frullanti, and A. Giordano. Differential gene expression analysis pipelines and bioinformatic tools for the identification of specific biomarkers: A review. *Computational and Structural Biotechnology Journal*, 23:1154–1168, 2024.
- [91] RunPod. Runpod: The cloud built for ai. <https://www.runpod.io/>, 2024. Accessed: 2024-06-06.
- [92] J. Saad-Falcon, A. G. Lafuente, S. Natarajan, N. Maru, H. Todorov, E. Guha, E. K. Buchanan, M. Chen, N. Guha, C. Ré, and A. Mirhoseini. Archon: An architecture search framework for inference-time techniques, 2024. URL <https://arxiv.org/abs/2409.15254>.
- [93] E. W. Sayers, E. E. Bolton, J. R. Brister, K. Canese, J. Chan, D. C. Comeau, R. Connor, K. Funk, C. Kelly, S. Kim, et al. Database resources of the national center for biotechnology information. *Nucleic Acids Research*, 50(D1):D20–D26, 2022. doi: 10.1093/nar/gkab1112.
- [94] S. Schmidgall and M. Moor. Agentrivx: Towards collaborative autonomous research. *arXiv preprint arXiv:2503.18102*, 2025.
- [95] S. Schmidgall, J. Achterberg, T. Knutins, L. Kirsch, R. Heira, C. R. Ahmad, and F. Iacobacci. Agent laboratory: Using llm agents as research assistants. *arXiv preprint arXiv:2501.04227*, 2025.
- [96] S. Schmidgall, Y. Su, Z. Wang, X. Sun, J. Wu, X. Yu, J. Liu, Z. Liu, and E. Barsoum. Agent labora-

- tory: Using llm agents as research assistants. *arXiv preprint arXiv:2501.04227*, 2025.
- [97] D. Shapiro, W. Li, M. Delaflor, and C. Toxtli. Conceptual framework for autonomous cognitive entities. *arXiv preprint arXiv: 2310.06775*, 2023.
- [98] M. Shen and Q. Yang. From mind to machine: The rise of manus ai as a fully autonomous digital agent, 2025.
- [99] C. Si, D. Yang, and T. Hashimoto. Can llms generate novel research ideas? a large-scale human study with 100+ nlp researchers. *arXiv preprint arXiv:2409.04109*, 2024.
- [100] K. Singhal, S. Azizi, T. Tu, S. S. Mahdavi, J. Wei, H. W. Chung, N. Scales, A. Tanwani, H. Cole-Lewis, S. Pfohl, et al. Large language models encode clinical knowledge. *Nature*, 620(7972): 172–180, 2023.
- [101] H. W. Sprueill, C. Edwards, M. V. Olarte, U. Sanyal, H. Ji, and S. Choudhury. Monte carlo thought search: Large language model querying for complex scientific reasoning in catalyst design. *arXiv preprint arXiv:2310.14420*, 2023.
- [102] H. W. Sprueill, C. Edwards, K. Agarwal, M. V. Olarte, U. Sanyal, C. Johnston, H. Liu, H. Ji, and S. Choudhury. Chemreasoner: Heuristic search over a large language model’s knowledge space using quantum-chemical feedback. *arXiv preprint arXiv:2402.10980*, 2024.
- [103] R. Stark, M. Grzelak, and J. Hadfield. Rna sequencing: the teenage years. *Nature Reviews Genetics*, 20(11):631–656, 2019. doi: 10.1038/s41576-019-0150-2.
- [104] A. Subramanian, P. Tamayo, V. K. Mootha, S. Mukherjee, B. L. Ebert, M. A. Gillette, A. Paulovich, S. L. Pomeroy, T. R. Golub, E. S. Lander, et al. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, 102(43):15545–15550, 2005.
- [105] T. R. Sumers, S. Yao, K. Narasimhan, and T. L. Griffiths. Cognitive architectures for language agents. *arXiv preprint arXiv: 2309.02427*, 2023.
- [106] K. Swanson, W. Wu, N. L. Bulaong, J. E. Pak, and J. Zou. The virtual lab: Ai agents design new sars-cov-2 nanobodies with experimental validation. *bioRxiv*, pages 2024–11, 2024.
- [107] Y. Talebirad and A. Nadiri. Multi-agent collaboration: Harnessing the power of intelligent llm agents. *arXiv preprint arXiv: 2306.03314*, 2023.
- [108] O. Targets. Open targets platform, 2025. URL <https://platform.opentargets.org/>. Accessed: February 24, 2025.
- [109] A. Thakur, G. Tsoukalas, Y. Wen, J. Xin, and S. Chaudhuri. An in-context learning agent for formal theorem-proving. *arXiv preprint arXiv:2310.04353*, 2023.
- [110] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.
- [111] K. Tomczak, P. Czerwińska, and M. Wiznerowicz. The cancer genome atlas (tcga): an immeasurable source of knowledge. *Contemporary Oncology (Poznan)*, 19(1A):A68–77, 2015. doi: 10.5114/wo.2014.47136.
- [112] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv: 2302.13971*, 2023.
- [113] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv: 2307.09288*, 2023.
- [114] K.-T. Tran, X. Che, J. Peng, X. Li, A. Manzoor, H. Yin, G. Li, and L. Liu. Multi-agent collaboration

mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

- [115] B. Wang, S. Min, X. Deng, J. Shen, Y. Wu, L. Zettlemoyer, and H. Sun. Towards understanding chain-of-thought prompting: An empirical study of what matters. *Annual Meeting of the Association for Computational Linguistics*, 2022. doi: 10.48550/arXiv.2212.10001.
- [116] H. Wang, B. Aragam, and E. P. Xing. Trade-offs of linear mixed models in genome-wide association studies. *Journal of Computational Biology*, 29(3):233–242, 2022.
- [117] K. Wang, Y. Lu, M. Santacroce, Y. Gong, C. Zhang, and Y. Shen. Adapting llm agents through communication. *arXiv preprint arXiv: 2310.01444*, 2023.
- [118] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, et al. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023.
- [119] P. Wang, Y. Yu, K. Chen, X. Zhan, and H. Wang. Large language model-based data science agent: A survey. In submission, 2025.
- [120] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
- [121] X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024.
- [122] Y. Wang and K.-A. LêCao. Managing batch effects in microbiome data. *Briefings in bioinformatics*, 21(6):1954–1970, 2020.
- [123] Y. Wang, Z. Jiang, Z. Chen, F. Yang, Y. Zhou, E. Cho, X. Fan, X. Huang, Y. Lu, and Y. Yang. Recmind: Large language model powered agent for recommendation. *arXiv preprint arXiv:2308.14296*, 2023.
- [124] Y. Wang, L. Yang, G. Li, M. Wang, and B. Aragam. Scoreflow: Mastering LLM agent workflows via score-based preference optimization. *arXiv preprint arXiv:2502.04306*, 2025.
- [125] Z. Wang, M. Gerstein, and M. Snyder. Rna-seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10(1):57–63, 2009. doi: 10.1038/nrg2484.
- [126] Z. Wang, S. Mao, W. Wu, T. Ge, F. Wei, and H. Ji. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*, 2023.
- [127] C.-H. Wei, A. Allot, K. Riehle, A. Milosavljevic, and Z. Lu. Gnorm2: an improved gene name recognition and normalization system. *Bioinformatics*, 39(10):btad599, 2023. doi: 10.1093/bioinformatics/btad599.
- [128] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.
- [129] J. N. Weinstein, E. A. Collisson, G. B. Mills, K. R. M. Shaw, B. A. Ozenberger, K. Ellrott, I. Shmulevich, C. Sander, and J. M. Stuart. The cancer genome atlas pan-cancer analysis project. *Nature genetics*, 45(10):1113–1120, 2013.
- [130] Y. Weng, M. Zhu, G. Bao, H. Zhang, J. Wang, Y. Zhang, and L. Yang. Cycleresearcher: Improving automated research via automated review. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=bjcsVLoHYs>.
- [131] C. Wu, I. Macleod, and A. I. Su. Biogps and mygene.info: organizing online, gene-centric information. *Nucleic Acids Research*, 41(Database issue):D561–D565, Jan 2013. doi: 10.1093/nar/gks1114.
- [132] Q. Wu, G. Bansal, J. Zhang, Y. Wu, S. Zhang, E. Zhu, B. Li, L. Jiang, X. Zhang, and C. Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.
- [133] T. T. Wu, Y. F. Chen, T. Hastie, E. Sobel, and K. Lange. Genome-wide association analysis by lasso penalized logistic regression. *Bioinformatics*, 25(6):714–721, 2009.
- [134] Z. Xi, S. Jin, Y. Zhou, R. Zheng, S. Gao, T. Gui, Q. Zhang, and X. Huang. Self-polish: Enhance

- reasoning in large language models via problem refinement. *arXiv preprint arXiv:2305.14497*, 2023.
- [135] Y. Xiao, J. Shen, X. Zhou, X. Fu, J. Liu, S. Zhang, Z. Zhou, X. Zhang, N. Liu, J. Liu, H. Zhang, and H. Chen. Cellagent: An llm-driven multi-agent framework for automated single-cell data analysis. *arXiv preprint arXiv:2407.09811*, 2024.
- [136] Y. Xiao, E. Sun, Y. Jin, Q. Wang, and W. Wang. Proteingpt: Multimodal llm for protein property prediction and structure understanding. *arXiv preprint arXiv:2408.11363*, 2024.
- [137] F. Xu, Q. Liu, Z. Liang, K. Chen, X. Zhang, H. Fei, Y. Yao, Z. Bao, Y. You, D. Cai, et al. Towards large reasoning models: A survey of reinforced reasoning with large language models. *arXiv preprint arXiv:2501.09686*, 2025.
- [138] Q. Xu, F. Hong, B. Li, C. Hu, Z. Chen, and J. Zhang. On the tool manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*, 2023.
- [139] Z. Xu, S. Shi, B. Hu, J. Yu, D. Li, M. Zhang, and Y. Wu. Towards reasoning in large language models via multi-agent peer review collaboration. *arXiv preprint arXiv: 2311.08152*, 2023.
- [140] E. Xue, Z. Huang, Y. Ji, and H. Wang. Improve: Iterative model pipeline refinement and optimization leveraging llm agents. *arXiv preprint arXiv:2502.18530*, 2025.
- [141] H. Yang, S. Yue, and Y. He. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv: 2306.02224*, 2023.
- [142] R. Yang, T. F. Tan, W. Lu, A. J. Thirunavukarasu, D. S. W. Ting, and N. Liu. Large language models in health care: Development, applications, and challenges. *Health Care Science*, 2(4):255–263, 2023.
- [143] Y. Yang, X. Li, Z. Wang, and J. Liu. Biologically weighted lasso: enhancing functional interpretability in gene expression data analysis. *Bioinformatics*, 40(10):btae605, 2024. doi: 10.1093/bioinformatics/btae605.
- [144] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.
- [145] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.
- [146] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. τ -bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL <https://arxiv.org/abs/2406.12045>.
- [147] J. Ye, G. Li, S. Gao, C. Huang, Y. Wu, S. Li, X. Fan, S. Dou, T. Ji, Q. Zhang, T. Gui, and X. Huang. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. *arXiv preprint arXiv:2401.00741*, 2024.
- [148] R. Ye, W. Zhang, J. Chen, M. Xue, Z. Gao, Y. Zhang, and H. Chen. Mas-gpt: Training llms to build llm-based multi-agent systems. *arXiv preprint arXiv:2503.03686*, 2025.
- [149] A. Yehudai, L. Huang, E. Arik, M. Yang, Z. Yang, N. Khattab, Y. Liang, and C. Yang. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*, 2025.
- [150] Z. Yin, Q. Sun, C. Chang, Q. Guo, J. Dai, X.-J. Huang, and X. Qiu. Exchange-of-thought: Enhancing large language model capabilities through cross-model communication. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15135–15153, 2023.
- [151] J. Yu, G. Pressoir, W. H. Briggs, I. Vroh Bi, M. Yamasaki, J. F. Doebley, M. D. McMullen, B. S. Gaut, D. M. Nielsen, J. B. Holland, et al. A unified mixed-model method for association mapping that accounts for multiple levels of relatedness. *Nature genetics*, 38(2):203–208, 2006.
- [152] Y. Yu, Y. Yu, K. Wei, H. Luo, and H. Wang. Sipdo: Closed-loop prompt optimization via synthetic data feedback. *arXiv preprint arXiv:2505.19514*, 2025.
- [153] G. Zhang, L. Niu, J. Fang, K. Wang, L. Bai, and X. Wang. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025.
- [154] J. Zhang, J. Xiang, Z. Yu, F. Teng, X. Chen, J. Chen, M. Zhuge, X. Cheng, S. Hong, J. Wang, B. Zheng, B. Liu, Y. Luo, and C. Wu. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.
- [155] P. Zhang, H. Jin, L. Hu, X. Li, L. Kang, M. Luo, Y. Song, and H. Wang. Revolve: Optimizing ai

systems by tracking response evolution in textual optimization. *arXiv preprint arXiv:2412.03092*, 2024.

- [156] S. Zhang, M. Yin, J. Zhang, J. Liu, Z. Han, J. Zhang, B. Li, C. Wang, H. Wang, Y. Chen, and Q. Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Forty-second International Conference on Machine Learning*, 2025. URL <https://openreview.net/forum?id=GazlTYxZss>.
- [157] T. Zhang, Z. Liu, Y. Xin, and Y. Jiao. Mooseagent: A llm based multi-agent framework for automating moose simulation. *arXiv preprint arXiv:2504.08621*, 2025.
- [158] Y. Zhang, G. Parmigiani, and W. E. Johnson. Overcoming the impacts of two-step batch effect correction on gene expression estimation and inference. *Biostatistics*, 24(2):502–522, 2020. doi: 10.1093/biostatistics/kxac041.
- [159] Y. Zhang, Y. Li, T. Zhao, K. Zhu, H. Wang, and N. Vasconcelos. Achilles heel of distributed multi-agent systems. *arXiv preprint arXiv:2504.07461*, 2025.
- [160] R. Zhao, X. Li, S. R. Joty, C. Qin, and L. Bing. Verify-and-edit: A knowledge-enhanced chain-of-thought framework. *Annual Meeting of the Association for Computational Linguistics*, 2023. doi: 10.48550/arXiv.2305.03268.
- [161] L. Zheng, R. Wang, and B. An. Synapse: Leveraging few-shot exemplars for human-level computer control. *arXiv preprint arXiv:2306.07863*, 2023.
- [162] J. Zhou, B. Xiang, X. Li, X. Liu, and X. Gao. Automated bioinformatics analysis via autoba. *arXiv preprint arXiv:2309.03242*, 2023.
- [163] P. Zhou, A. Madaan, S. P. Potharaju, A. Gupta, K. R. McKee, A. Holtzman, J. Pujara, X. Ren, S. Mishra, A. Nematzadeh, S. Upadhyay, and M. Faruqui. How far are large language models from agents with theory-of-mind? *arXiv preprint arXiv: 2310.03051*, 2023.
- [164] M. Zhu, Y. Weng, L. Yang, and Y. Zhang. Deepreview: Improving llm-based paper review with human-like deep thinking process, 2025. URL <https://arxiv.org/abs/2503.08569>.
- [165] M. Zhuge, W. Wang, L. Kirsch, F. Faccio, D. Khizbulin, and J. Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *ICML*. OpenReview.net, 2024.
- [166] T. Zhuge, Y. Zhao, and H. Li. Orchestra: Orchestrating multiple llm agents through model heterogeneity and role specialization. *arXiv preprint arXiv:2501.01234*, 2025.
- [167] T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, C. Fang, Y. Gao, J. Gao, Z. Wu, L. Xia, J. An, et al. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 2024.
- [168] Y. Zimmermann, B. Blaiszik, I. Foster, A. Stevens, D. Audus, J. Montoya, M. Torresi, J. Dagdelen, A. Rosen, K. Cruse, et al. 34 examples of llm applications in materials science and chemistry: Towards automation, assistants, agents, and accelerated scientific discovery. *arXiv preprint arXiv:2505.03049*, 2025.

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:
```

```
    return None
```

This fundamental misinterpretation reveals a critical failure to understand disease biology. Liver cirrhosis accounts for approximately 90% of portal hypertension cases in Western countries and represents the most common etiology of this condition. By misclassifying LC samples as controls, Biomni not only demonstrates inadequate biomedical knowledge but also dramatically alters the statistical composition of the dataset. The incorrect classification creates a dataset with only 18 cases (INCPH alone) versus 36 controls (HNL and LC combined), while the scientifically correct classification would yield 40 cases (both INCPH and LC) versus merely 14 controls (HNL only). This reversal in the case-control ratio fundamentally compromises any downstream analysis. Rather than identifying genes associated with portal hypertension, the misclassified analysis would instead detect genes that distinguish INCPH from a heterogeneous mixture of healthy and cirrhotic livers, rendering the results biologically meaningless.

The error persists even under the detailed prompt condition, where explicit guidelines and domain knowledge are provided. In the general prompt condition, Biomni compounds the misclassification by entirely excluding LC samples from the analysis:

```
# From Biomni's general prompt processing
trait_mapping = {
    'HNL': 0,      # Control - histologically normal livers
    'INCPH': 1,    # Disease - idiopathic non-cirrhotic portal hypertension
    'LC': 'exclude' # Exclude liver cirrhosis samples <--}
}

# Filter samples and create sample metadata
final_sample_metadata = {}
for sample_id, metadata in sample_metadata.items():
    liver_group = metadata['liver_group']
    if liver_group != 'LC': # Exclude LC samples <-->
        final_sample_metadata[sample_id] = {
            'INCPH': trait_mapping[liver_group],
            'liver_group': liver_group
        }

print(f"Samples after filtering (excluding LC): {len(final_sample_metadata)}")
# Output: Samples after filtering (excluding LC): 32
```

This exclusion strategy eliminates 22 LC samples from the original 54, representing a 40% data loss that severely compromises statistical power. The decision to exclude rather than correctly classify LC samples suggests a fundamental misunderstanding of the biological relationship between cirrhosis and portal hypertension.

Beyond the trait misclassification, Biomni exhibits several additional processing errors that compound the analysis invalidity. The agent fails to normalize gene symbols using standardized databases, a critical step performed by human experts to ensure cross-dataset compatibility. While the human expert code systematically applies gene synonym mapping to standardize identifiers, Biomni retains the raw gene symbols from the platform annotation, potentially introducing inconsistencies in downstream analyses. Furthermore, Biomni omits systematic bias detection and correction procedures. The human expert implementation includes explicit checks for biased features and removes them before final analysis, whereas Biomni proceeds without such quality control measures.

These cascading errors, from fundamental biological misinterpretation to technical processing omissions, illustrate that even specialized biomedical tools cannot compensate for the lack of precise procedural control required for valid scientific analysis. The inability to recognize that different diseases can manifest the same clinical phenotype, combined with the failure to implement standard bioinformatics quality control procedures, renders the autonomous agent's output scientifically unreliable.

A.3 AutoGen Examples

AutoGen, a popular multi-agent conversation framework developed by Microsoft [132], demonstrates fundamental limitations when applied to gene expression analysis. While designed to facilitate autonomous agent collaboration through conversational patterns, AutoGen struggles with the precise procedural control required for bioinformatics workflows. The following examples illustrate how agent miscommunication and lack of domain understanding lead to critical analysis errors.

A.3.1 GSE77627: Agent Coordination Failures in Trait Classification

Under the general prompt condition, AutoGen attempts to coordinate multiple agents for the portal hypertension dataset analysis but fails due to miscommunication between agents about the biological meaning of liver conditions:

```
# AutoGen multi-agent setup
import autogen

# Define specialized agents
bioinformatics_agent = autogen.AssistantAgent(
    name="BioinformaticsExpert",
    system_message="You are a bioinformatics expert. Analyze gene data."
)

clinical_agent = autogen.AssistantAgent(
    name="ClinicalExpert",
    system_message="You are a clinical expert. Handle phenotype data."
)

# Clinical agent's trait interpretation
def analyze_clinical_groups(data):
    """Clinical agent's analysis of liver groups"""
    groups = data['liver_group'].unique()
    # Agent message: "I see three groups: HNL, INCPH, and LC"
    # Agent message: "INCPH is clearly the disease of interest"
    # Agent message: "LC might be a different condition, treating as control" <-->

    trait_mapping = {
        'HNL': 0,      # Healthy controls
        'INCPH': 1,    # Disease cases
        'LC': 0        # Misclassified as control <--
    }
    return trait_mapping

# Bioinformatics agent accepts without domain verification
def process_with_mapping(expression_data, trait_mapping):
    """Bioinformatics agent processes based on clinical interpretation"""
    # Agent message: "Received trait mapping from clinical expert"
    # Agent message: "Processing expression data with 2-class comparison"
    # No validation of biological correctness <-->
    return apply_trait_classification(expression_data, trait_mapping)
```

This example reveals a critical failure in agent coordination. The clinical agent misinterprets liver cirrhosis (LC) as a control condition, and the bioinformatics agent accepts this classification without question. In AutoGen's conversational paradigm, agents tend to defer to each other's "expertise" without implementing cross-validation mechanisms. This leads to the propagation of fundamental biological errors through the analysis pipeline.

Under the detailed prompt condition, AutoGen's conversation-based approach leads to verbose but ineffective deliberation:

```
# AutoGen conversation with detailed guidelines
user_proxy = autogen.UserProxyAgent(
    name="User",
    human_input_mode="NEVER",
    code_execution_config={"work_dir": "analysis"}
)

# Extended conversation pattern
conversation_history = []

# Round 1: Initial interpretation
clinical_agent: "Based on the guidelines, I need to classify liver groups."
clinical_agent: "INCPH stands for Idiopathic Non-Cirrhotic Portal Hypertension."
clinical_agent: "This is clearly our case group (trait = 1)."

# Round 2: Misguided reasoning
clinical_agent: "LC stands for Liver Cirrhosis."
clinical_agent: "Since it's 'cirrhotic' and INCPh is 'non-cirrhotic',"
clinical_agent: "they seem to be opposite conditions." <-->
clinical_agent: "Therefore, LC should be grouped with HNL as controls."
```

```

# Round 3: Bioinformatics agent agrees without domain knowledge
bioinformatics_agent: "I accept the clinical classification."
bioinformatics_agent: "Proceeding with binary trait: INCPh=1, others=0.."

# No agent questions the biological validity <--
```

The conversational approach, while appearing thorough, actually amplifies the initial misunderstanding through repeated affirmation between agents. Neither agent possesses the domain knowledge to recognize that both INCPh and LC cause portal hypertension through different mechanisms.

A.3.2 GSE117261: Task Decomposition Without Domain Context

For the GSE117261 dataset, AutoGen's approach to task decomposition fails to identify the critical probe-to-gene mapping requirement:

```

# AutoGen task decomposition
planner_agent = autogen.AssistantAgent(
    name="Planner",
    system_message="Break down data preprocessing into subtasks."
)

# Planner's task breakdown
task_list = planner_agent.generate_reply(
    messages=[{
        "content": "Preprocess GEO series matrix file GSE117261",
        "role": "user"
    }]
)

# Generated task list (missing critical steps):
# 1. Load series matrix file
# 2. Extract sample metadata
# 3. Extract expression values
# 4. Merge metadata with expression
# 5. Save processed dataset

# Execution by data processing agent
def execute_preprocessing_plan(tasks):
    # Task 3: Extract expression values
    expression_df = pd.read_csv(matrix_file, sep='\t', comment='!')
    expression_df = expression_df.set_index('ID_REF') # Probe IDs <--
```

Task 4: Merge (probe IDs used directly as features)

```
    final_df = pd.concat([metadata_df, expression_df.T], axis=1) <--
```

return final_df # Contains probe IDs, not gene symbols

The planner agent, lacking bioinformatics knowledge, fails to include probe-to-gene mapping in the task decomposition. Subsequent agents execute the incomplete plan without recognizing the omission. This demonstrates how AutoGen's strength in task orchestration becomes a weakness when critical domain-specific steps are not identified during planning.

A.4 MetaGPT Examples

MetaGPT, a multi-agent framework emphasizing structured role-based collaboration [44], exhibits similar fundamental limitations despite its sophisticated role definitions and standardized operating procedures. The framework's emphasis on software engineering patterns fails to translate effectively to scientific data analysis requirements.

A.4.1 GSE256539: Role-Based Misinterpretation of Clinical Encoding

MetaGPT's role-based architecture assigns specialized responsibilities to different agents. However, this specialization leads to fragmented understanding when processing the GSE256539 IPAH dataset:

```

# MetaGPT role definitions
class DataAnalyst(Role):
    """Analyzes data patterns and structures"""
```

```

def _observe(self):
    return [UserRequirement]

def _act(self) -> dict:
    # Analyze sample characteristics
    sample_chars = self._analyze_characteristics()

    # DataAnalyst observation: "Samples have individual IDs"
    # Example: "individual: AH014", "individual: UC003"
    # DataAnalyst conclusion: "No clear trait information" <-->

    return {
        "observation": "Dataset contains individual identifiers only",
        "trait_found": False,
        "recommendation": "Cannot extract hypertension trait" <-->
    }

class DomainExpert(Role):
    """Provides domain-specific interpretations"""

    def _act_on_analysis(self, analyst_report: dict) -> dict:
        if not analyst_report["trait_found"]:
            # DomainExpert accepts analyst conclusion without examination
            return {
                "interpretation": "No trait data available",
                "action": "Skip trait extraction" <-->
            }

```

The rigid role boundaries prevent effective information flow. The DataAnalyst, focused on data structure, fails to recognize the clinical significance of ID prefixes. The DomainExpert, receiving only the analyst's conclusion, never examines the actual data to discover that 'AH' likely stands for Arterial Hypertension. This demonstrates how role-based decomposition can create blind spots in analysis.

Under detailed prompt conditions, MetaGPT's structured approach produces extensive documentation but still misses the critical pattern:

```

class ClinicalDataProcessor(Action):
    """Structured action for clinical data processing"""

    async def run(self, data: pd.DataFrame) -> ProcessingResult:
        # Generate structured analysis document
        analysis_doc = Document()

        # Section 1: Data Overview
        analysis_doc.add_section("Data Characteristics")
        for col in data.columns:
            if 'individual' in col:
                unique_vals = data[col].unique()[:5]
                analysis_doc.add_item(f"Column {col}: {unique_vals}")
                # Output: "Column ch1: ['individual: AH014', 'individual: UC003', ...]"

        # Section 2: Trait Identification Strategy
        analysis_doc.add_section("Trait Search Strategy")
        search_patterns = ['hypertension', 'ipah', 'pah', 'control', 'disease']

        # Searches for keywords but ignores ID patterns <-->
        matches_found = False
        for pattern in search_patterns:
            if any(pattern in str(val).lower() for val in data.values.flat):
                matches_found = True

        # Section 3: Conclusion
        if not matches_found:
            analysis_doc.add_conclusion(
                "No trait indicators found in dataset" <-->
            )

    return ProcessingResult(success=False, trait_mapping=None)

```

Despite generating comprehensive documentation, MetaGPT's structured approach focuses on keyword searching rather than pattern recognition. The framework's emphasis on standardized proce-

dures prevents the adaptive reasoning needed to recognize that ID prefixes encode phenotype information.

A.4.2 GSE77627: Procedural Rigidity in Biological Classification

When processing the portal hypertension dataset, MetaGPT's standardized operating procedures lead to systematic misclassification:

```
class StandardOperatingProcedure:
    """SOP for disease trait classification"""

    @staticmethod
    def classify_disease_samples(disease_name: str, sample_groups: list) -> dict:
        """
        Standard procedure for binary classification:
        1. Identify primary disease group
        2. Classify all others as controls
        """

        classification = {}
        primary_disease = None

        # Step 1: Find exact disease name match
        for group in sample_groups:
            if disease_name.lower() in group.lower():
                primary_disease = group
                classification[group] = 1
                break

        # Step 2: Classify remaining as controls
        for group in sample_groups:
            if group not in classification:
                classification[group] = 0  <-->

        return classification

# Execution for portal hypertension
sample_groups = ['HNL', 'INCPH', 'LC']
disease_name = 'Hypertension'

# SOP application results:
# - No exact match for "Hypertension" in group names
# - Falls back to INCPH as primary (contains "Hypertension")
# - Classifies LC as control  <-->

trait_mapping = {
    'INCPH': 1,  # Matched "Hypertension" substring
    'HNL': 0,    # Classified as control
    'LC': 0      # Incorrectly classified as control  <-->
}
```

MetaGPT's rigid adherence to standardized procedures prevents recognition of domain-specific relationships. The SOP's simplistic "one disease, others control" rule cannot accommodate the biological reality that multiple conditions can manifest the same phenotype. Even when the Architect agent designs the workflow and the Engineer agent implements it perfectly according to specifications, the fundamental misunderstanding persists because it is encoded in the procedural logic.

These examples from AutoGen and MetaGPT reinforce our findings about the fundamental limitations of current autonomous agent frameworks. Despite sophisticated multi-agent architectures, conversational capabilities, and structured procedures, these systems lack the domain understanding and adaptive reasoning required for scientific data analysis. Their failures highlight the necessity of incorporating precise procedural control and domain expertise directly into agent architectures rather than relying on emergent behaviors from general-purpose frameworks.

A.5 Claude Code Examples

Claude Code, arguably today's best general-purpose CLI-based agent, demonstrates similar fundamental limitations. This proprietary agent excels at general software development tasks but struggles with

domain-specific scientific data analysis. The following examples illustrate how lack of domain knowledge and procedural control leads to critical errors in both prompt conditions.

A.5.1 GSE77627: Misclassification of Portal Hypertension Subtypes

Under the general prompt condition, Claude Code implements a three-class classification scheme that fundamentally misunderstands the biological relationship between disease subtypes:

```
# Claude Code's trait conversion logic
def preprocess_GSE77627():
    # ... earlier code omitted ...

    # Convert liver groups to binary/categorical encoding
    # HNL (healthy normal liver) = 0 (control)
    # INCPh (idiopathic non-cirrhotic portal hypertension) = 1 (case)
    # LC (liver cirrhosis) = 2 (another condition) <-->
    group_mapping = {'HNL': 0, 'INCPh': 1, 'LC': 2}
    metadata_df['trait'] = metadata_df['liver_group'].map(group_mapping)

    print("Sample distribution:")
    print(metadata_df['liver_group'].value_counts())

    # Note: Age and Gender are not available in this dataset
    print(f"\nFinal dataset shape: {final_df.shape}")
    print(f"Columns: trait + {final_df.shape[1]-1} gene expression features")
    print(f"Trait column: 0=HNL(control), 1=INCPh(case), 2=LC") <-->
```

This multi-class formulation reveals a critical failure to recognize that both INCPh and LC represent different etiologies of portal hypertension—the trait under investigation. Liver cirrhosis is the leading cause of portal hypertension, accounting for approximately 90% of cases. By assigning LC to a separate class (2) rather than grouping it with INCPh as cases (1), Claude Code creates a biologically meaningless classification that cannot identify genes associated with portal hypertension.

The error persists even under the detailed prompt condition, where explicit guidelines are provided. Claude Code's implementation still misclassifies LC samples:

```
def convert_trait(value):
    """Convert trait values to binary format"""
    value_str = str(value).lower()

    # Extract the liver group information
    if 'incph' in value_str:
        return 1 # INCPh as the target condition
    elif 'lc' in value_str or 'cirrhosis' in value_str:
        return 0 # Cirrhosis as control/comparison <-->
    elif 'hnl' in value_str or 'normal' in value_str:
        return 0 # Normal liver as control
    else:
        return None
```

Despite the detailed guidelines emphasizing domain knowledge incorporation, Claude Code treats cirrhosis as a control condition rather than recognizing it as the most common cause of portal hypertension. This fundamental misunderstanding demonstrates that even sophisticated code generation capabilities cannot compensate for the absence of biomedical reasoning in critical decision points.

A.5.2 GSE117261: Failure to Map Probe Identifiers

For the GSE117261 dataset, Claude Code exhibits a different but equally critical failure—the complete absence of probe-to-gene mapping. Under both prompt conditions, the agent produces code that retains raw probe identifiers instead of mapping them to standardized gene symbols:

```
def create_final_dataset():
    """Create the final preprocessed dataset"""
    # ... metadata extraction code ...

    # Transpose expression data to have samples as rows
    print("\nTransposing expression data...")
    expression_t = expression_data.set_index('ID_REF').T <-->
    expression_t.index.name = 'sample_id'
```

```

# Merge metadata with expression data
print("Merging metadata with expression data...")
final_df = sample_df.set_index('sample_id').join(expression_t)    <--

print(f"Final dataset shape: {final_df.shape}")
print(f"Columns: {list(final_df.columns[:10])}... (showing first 10)")

# Save the processed dataset
output_file = "/home/techt/Desktop/gene/gene_cc/GSE117261_processed.csv"
print(f"\nSaving processed dataset to {output_file}...")
final_df.to_csv(output_file)

```

The code directly uses the raw probe IDs (e.g., '7892501', '7892502') as column names without any attempt to map them to gene symbols. This omission renders the dataset incompatible with downstream analyses that require standardized gene identifiers. In contrast, the human expert implementation performs comprehensive probe mapping:

```

# Human expert implementation (excerpt)
# Extract gene annotation from SOFT file
gene_annotation = get_gene_annotation(soft_file_path)

# Map probe IDs to gene symbols
prob_col = 'ID'
gene_col = 'gene_assignment'
mapping_data = get_gene_mapping(gene_annotation, prob_col, gene_col)
gene_data = apply_gene_mapping(expression_df=genetic_data,
                               mapping_df=mapping_data)

# Normalize gene symbols using NCBI Gene database
gene_data_normalized = normalize_gene_symbols_in_index(gene_data)

```

The absence of these critical preprocessing steps—annotation extraction, probe mapping, and symbol normalization—demonstrates Claude Code’s inability to recognize and implement essential bioinformatics workflows autonomously. Even when provided with detailed guidelines that explicitly mention gene mapping requirements, the agent fails to implement this fundamental preprocessing step, producing datasets that cannot be used for meaningful biological analysis.

These examples from Claude Code reinforce our findings about the fundamental limitations of current autonomous agents in scientific computing. Despite advanced code generation capabilities, these systems lack the domain understanding and procedural control mechanisms necessary to ensure biologically valid analyses.

A.6 Cursor Examples

Cursor, one of the current best agentic code editors, demonstrates similar fundamental limitations when applied to gene expression analysis. While excelling at general programming tasks through its integrated AI assistance, Cursor fails to implement critical bioinformatics workflows that require domain-specific knowledge. The following examples illustrate how these failures manifest differently from Claude Code while reinforcing the same underlying limitations.

A.6.1 GSE117261: Complete Absence of Probe-to-Gene Mapping

Under the general prompt condition, Cursor exhibits a critical failure in processing the GSE117261 dataset by entirely omitting the probe-to-gene mapping step. The dataset contains Affymetrix microarray data with numeric probe identifiers that must be mapped to standardized gene symbols for meaningful analysis.

Cursor’s implementation demonstrates a fundamental misunderstanding of gene expression data requirements:

```

def get_gene_symbols(expr_df: pd.DataFrame) -> pd.DataFrame:
    """
    Convert probe IDs to gene symbols. For this dataset, we'll use the probe IDs as gene
    identifiers
    since gene symbol mapping would require additional annotation files.

Args:

```

```

expr_df: Expression dataframe with probe IDs as index

Returns:
    Expression dataframe with gene symbols (or probe IDs) as column names
"""
print("Using probe IDs as gene identifiers (gene symbol mapping not available)") <-->

# Transpose so samples are rows and genes are columns
expr_df_t = expr_df.T

# Clean up column names (remove quotes if any)
expr_df_t.columns = [str(col).strip('') for col in expr_df_t.columns]

return expr_df_t

```

This implementation reveals a critical gap in understanding bioinformatics data processing. By declaring "gene symbol mapping not available" and proceeding with raw probe IDs, Cursor creates a dataset that is fundamentally incompatible with downstream analyses. The resulting data contains columns labeled with numeric identifiers like '7892501', '7892502' rather than meaningful gene symbols like 'TP53' or 'EGFR'.

In contrast, the human expert implementation performs comprehensive probe mapping:

```

# Human expert implementation (excerpt)
# Step 1: Extract gene annotation from SOFT file
gene_annotation = get_gene_annotation(soft_file_path)

# Step 2: Create mapping from probe IDs to gene symbols
prob_col = 'ID'
gene_col = 'gene_assignment'
mapping_data = get_gene_mapping(gene_annotation, prob_col, gene_col)

# Step 3: Apply mapping to convert probe-level to gene-level data
gene_data = apply_gene_mapping(expression_df=genetic_data,
                               mapping_df=mapping_data)

# Step 4: Normalize gene symbols using NCBI Gene database
gene_data_normalized = normalize_gene_symbols_in_index(gene_data)

```

The consequences of this omission cascade through the analysis pipeline. Without proper gene symbols, the dataset cannot be integrated with other genomic resources, compared across studies, or interpreted in biological context. This represents not merely a technical oversight but a fundamental failure to understand the requirements of genomic data analysis, where standardized gene identifiers are essential for scientific validity.

A.6.2 GSE256539: Misinterpretation of Clinical Phenotype Encoding

Under the detailed prompt condition, Cursor demonstrates a different but equally consequential failure in processing the GSE256539 dataset. This study compares Idiopathic Pulmonary Arterial Hypertension (IPAH) patients with controls, where the phenotype information is encoded in individual ID prefixes rather than explicit labels.

Cursor implements a generic trait conversion function that searches for obvious keywords:

```

def convert_trait(value):
    """Convert trait value to binary (0/1) or continuous"""
    if pd.isna(value) or value is None:
        return None

    value_str = str(value).lower()

    # Extract value after colon if present
    if ':' in value_str:
        value_str = value_str.split(':', 1)[1].strip()

    # Binary conversion for disease status
    if any(keyword in value_str for keyword in ['control', 'normal', 'healthy', 'negative']):
        <-->
        return 0
    elif any(keyword in value_str for keyword in ['case', 'disease', 'cancer', 'tumor', 'positive', 'affected']):
        <-->

```

```

    return 1

    # Try to convert to numeric for continuous traits
    try:
        return float(value_str)
    except:
        return None

```

This simplistic keyword-based approach fails entirely for the GSE256539 dataset, where sample characteristics contain individual IDs like 'individual: AH014' or 'individual: UC003'. None of these IDs contain the expected keywords, resulting in all samples being classified as None and subsequently dropped from the analysis.

The human expert implementation recognizes the sophisticated encoding scheme:

```

def convert_trait(value: str) -> Optional[int]:
    """
    Converts individual ID to a binary trait value (IPAH vs Control).
    - 1: IPAH patient (case)
    - 0: Control patient
    The distinction is inferred from the prefix of the individual ID string.
    e.g., 'individual: AH014'.
    """
    try:
        id_part = value.split(':')[1].strip()
        prefix = id_part[:2]
        # Based on the study design (IPAH vs. control) and observed prefixes,
        # 'AH' (Arterial Hypertension), 'VA', 'BA' are inferred as cases.
        # 'UC' (Unaffected Control), 'UM' (Unaffected Male), 'CC' (Control Case),
        # and 'UA' (Unaffected) are inferred as controls.
        if prefix in ['AH', 'VA', 'BA']:
            return 1 # Case
        elif prefix in ['UC', 'UM', 'CC', 'UA']:
            return 0 # Control
        else:
            return None
    except (IndexError, AttributeError):
        return None

```

This example illustrates a critical limitation: even when provided with detailed guidelines through our Action Unit specifications, Cursor cannot adapt its generic pattern-matching approach to the specific encoding schemes used in real genomic datasets. The agent fails to examine the actual data structure, recognize the pattern in individual IDs, or infer the biological meaning from the prefixes. This results in complete data loss as no samples can be properly classified.

The failure is particularly revealing because it occurs despite the detailed prompt condition providing explicit instructions about careful data examination and domain knowledge incorporation. Cursor's inability to move beyond generic keyword matching to context-specific pattern recognition demonstrates the absence of adaptive reasoning required for scientific data analysis.

A.7 Manus Examples

Manus, a recently proposed multi-agent system that claims general-purpose task automation capabilities [98], demonstrates fundamental failures when applied to gene expression analysis. While designed to orchestrate multiple specialized agents for complex workflows, Manus lacks the domain understanding and procedural control necessary for bioinformatics data processing. The following examples illustrate critical errors in both prompt conditions.

A.7.1 GSE117261: Complete Failure in Probe-to-Gene Mapping

Under the general prompt condition, Manus exhibits multiple critical failures in processing the GSE117261 dataset, most notably the complete absence of probe-to-gene mapping. This dataset contains Affymetrix microarray data where expression values are indexed by numeric probe identifiers that must be mapped to standardized gene symbols.

Manus's implementation reveals a fundamental misunderstanding of gene expression data structure:

```

def process_expression_file(filepath):

```

```

with open(filepath, 'r') as f:
    lines = f.readlines()

    # Find data start
    start_index = -1
    for i, line in enumerate(lines):
        if line.strip() == '!series_matrix_table_begin':
            start_index = i + 1
            break

    # Read expression data
    df = pd.read_csv(filepath, sep='\t', skiprows=start_index, index_col=0)

    # Transpose so samples are columns
    df = df.transpose()
    df.index.name = 'Sample_ID'

    return df # Returns data with probe IDs as columns <-->

# In main execution:
# Process expression data
expression_df = process_expression_file(series_matrix_path)

# Merge dataframes - directly uses probe IDs as features
final_df = pd.merge(metadata_df, expression_df,
                    left_index=True, right_index=True, how='outer') <--

```

The code directly merges clinical metadata with expression data that retains numeric probe identifiers (e.g., '7892501', '7892502') as column names. This fundamental omission renders the dataset scientifically unusable, as downstream analyses require standardized gene symbols for biological interpretation and cross-study comparison.

Beyond the probe mapping failure, Manus demonstrates additional critical errors in clinical data processing:

```

# Incorrect clinical characteristic extraction
if 'clinical_group' in data['characteristics']: # Wrong key <--
    trait = data['characteristics']['clinical_group']
    if trait == 'all PAH': # Incomplete mapping <--
        trait = 1
    elif trait == 'FD':
        trait = 0
    # Missing: other PAH subtypes are ignored <--

```

Manus searches for a non-existent 'clinical_group' key when the actual characteristic is 'pah_subtype'. Furthermore, the trait mapping logic fails to recognize that all PAH (Pulmonary Arterial Hypertension) subtypes should be classified as cases, not just 'all PAH'. This results in data loss and misclassification of disease samples.

In contrast, the human expert implementation performs comprehensive processing:

```

# Human expert: Proper probe-to-gene mapping
# Step 1: Extract gene annotation from SOFT file
gene_annotation = get_gene_annotation(soft_file_path)

# Step 2: Create mapping from probe IDs to gene symbols
prob_col = 'ID'
gene_col = 'gene_assignment'
mapping_data = get_gene_mapping(gene_annotation, prob_col, gene_col)

# Step 3: Apply mapping to convert probe-level to gene-level data
gene_data = apply_gene_mapping(expression_df=genetic_data,
                               mapping_df=mapping_data)

# Step 4: Normalize gene symbols using NCBI Gene database
gene_data_normalized = normalize_gene_symbols_in_index(gene_data)

```

The absence of these critical bioinformatics steps demonstrates Manus's inability to recognize and implement essential data processing requirements autonomously, producing datasets that cannot support meaningful biological analysis.

A.7.2 GSE77627: Fundamental Misunderstanding of Disease Biology

Under the detailed prompt condition with explicit guidelines, Manus demonstrates an even more concerning failure—fundamental misunderstanding of disease biology that persists despite detailed instructions. Processing the GSE77627 portal hypertension dataset, Manus implements biologically incorrect trait classification:

```
# Manus's trait conversion with detailed instructions
def convert_trait(value):
    value = value.split(":")[-1].strip().strip("\\")

    if value == "INCPH":
        return 1
    elif value == "LC" or value == "HNL": # Critical biological error <--
        return 0
    return None

# Agent's note reveals the misunderstanding
validate_and_save_cohort_info(
    is_gene_available,
    trait_available,
    notes="Trait 'liver group' is available and converted to binary " +
        "(INCPH=1, LC/HNL=0). Age and Gender data are not explicitly available."
) <--
```

This implementation classifies liver cirrhosis (LC) as a control condition alongside healthy livers (HNL), demonstrating a critical failure to understand portal hypertension etiology. Liver cirrhosis accounts for approximately 90% of portal hypertension cases in Western countries and is the most common cause of this condition. By misclassifying LC samples as controls, Manus creates a dataset where the majority of disease cases are labeled as healthy controls.

The correct biological classification, implemented by the human expert, recognizes both disease subtypes:

```
# Human expert: Biologically correct classification
def convert_trait(value: str) -> Optional[int]:
    """Converts liver group status to a binary value for Hypertension."""
    try:
        group = value.split(':')[1].strip().upper()
        if group in ['INCPH', 'LC']: # Both are portal hypertension
            return 1
        elif group == 'HNL': # Only healthy livers are controls
            return 0
        else:
            return None
    except (IndexError, AttributeError):
        return None
```

The impact of this misclassification is severe: Manus's approach yields only 18 cases (INCPH) versus 36 controls (HNL and LC combined), while the correct classification produces 40 cases versus 14 controls. This reversal of the case-control ratio fundamentally alters the dataset's statistical properties and would lead to identification of genes that distinguish INCPH from a heterogeneous mixture of healthy and cirrhotic livers, rather than genes associated with portal hypertension.

Furthermore, Manus attempts probe-to-gene mapping but implements it incorrectly:

```
# Manus attempts mapping but with errors
def map_probes_to_genes(series_matrix_df, annotation_path, output_path):
    annotation_df = pd.read_csv(annotation_path, sep='\t', comment='#')
    probe_to_gene_map = annotation_df.set_index('ID')['Symbol'].to_dict()
    series_matrix_df['Gene_Symbol'] = series_matrix_df['ID_REF'].map(probe_to_gene_map)
    # ... saves to file but doesn't properly integrate <--

    # Later in merging:
    expression_df_transposed = mapped_df_for_analysis.set_index("Gene_Symbol")
        .drop("ID_REF", axis=1).T <--
    # This assumes Gene_Symbol column exists and handles duplicates poorly
```

While Manus attempts probe mapping, the implementation lacks proper handling of many-to-one probe-gene relationships and fails to integrate the mapped data correctly into the final dataset. The agent also omits critical quality control steps such as gene symbol normalization and systematic bias detection.

These examples from Manus reinforce our findings about fundamental limitations in current autonomous agents. Despite claims of general-purpose automation and multi-agent orchestration capabilities, Manus fails to implement basic bioinformatics workflows correctly. The persistence of biological misunderstandings even under detailed prompt conditions demonstrates that sophisticated agent architectures cannot compensate for the absence of domain understanding and precise procedural control required for scientific data analysis.

B Communication Protocols and Messaging Mechanisms

This section provides technical details about the communication protocols that enable agent coordination in GenoMAS. The system employs a typed message-passing architecture that facilitates structured interactions between agents while maintaining modularity and preventing circular dependencies.

B.1 Message Type Hierarchy

GenoMAS defines a comprehensive set of message types that govern agent interactions. These types form a structured hierarchy organized into three categories that reflect the fundamental communication patterns in the system.

Request Messages The system uses five types of request messages to initiate agent actions:

- TASK_REQUEST: Initiates a new task assignment from the PI agent
- CODE_WRITING_REQUEST: Requests code generation for a specific action unit
- CODE REVIEW REQUEST: Requests review of generated code
- CODE REVISION REQUEST: Requests revision based on review feedback
- PLANNING REQUEST: Requests selection of the next action unit

Response Messages Each request type has a corresponding response type:

- TASK_RESPONSE: Confirms task assignment
- CODE_WRITING_RESPONSE: Contains generated code
- CODE REVIEW_RESPONSE: Contains approval/rejection decision and feedback
- CODE REVISION_RESPONSE: Contains revised code
- PLANNING_RESPONSE: Contains selected action unit and reasoning

System Messages Special messages for system-level events:

- TIMEOUT: Indicates that an agent has exceeded its time limit

The message types are organized to maintain clear request-response symmetry, with automatic mapping between corresponding pairs to ensure protocol consistency.

B.2 Message Structure

Each message contains four essential components: (1) the sender's **role**, (2) the message **type** from the hierarchy above, (3) the **content** payload (task instructions, code, feedback, etc.), and (4) the **target roles** specifying intended recipients. This structure enables both unicast and multicast communication patterns, allowing messages to target single agents or multiple agents as needed.

B.3 Topology-Based Message Routing

GenoMAS implements a topology-based routing system that defines which agents can communicate with each other. This topology forms the backbone of the system's communication infrastructure, ensuring that messages flow along predefined paths that reflect the logical relationships between agents.

The topology is encoded as a directed graph where each agent maintains a set of other agents it can communicate with:

```
PI Agent: {PI Agent, GEO Agent, TCGA Agent,  
Statistician Agent, Code Reviewer, Domain Expert}  
  
GEO Agent: {PI Agent, GEO Agent, Code Reviewer, Domain Expert}  
  
TCGA Agent: {PI Agent, TCGA Agent, Code Reviewer, Domain Expert}  
  
Statistician Agent: {PI Agent, Statistician Agent, Code Reviewer}  
  
Code Reviewer: {PI Agent, GEO Agent, TCGA Agent,  
Statistician Agent}  
  
Domain Expert: {PI Agent, GEO Agent, TCGA Agent}
```

This topology prevents circular dependencies and infinite message loops by carefully controlling communication paths. For example, the Domain Expert cannot directly communicate with the Statistician Agent, reflecting their distinct responsibilities. While agents can send messages to themselves (enabling self-planning), the overall structure prevents multi-agent cycles.

The environment enforces these constraints by filtering messages based on the topology, ensuring only authorized communication occurs.

B.4 Asynchronous Message Queue Processing

The environment orchestrates agent interactions through a FIFO message queue that processes messages asynchronously. This design decouples agent execution, allowing each agent to operate at its own pace while maintaining overall system coordination.

The processing loop extracts messages from the queue, validates receivers against topology constraints, and invokes each valid receiver's `act()` method. New messages generated during processing are appended to the queue, creating a dynamic flow of interactions. This asynchronous design allows agents to operate independently while maintaining coordinated behavior, and scales naturally as new agent types are added to the system.

B.5 Agent Communication Patterns

Programming agents orchestrate complex workflows through carefully designed communication patterns implemented in the `act()` method. These patterns enable agents to collaborate effectively while maintaining clear separation of concerns.

Task Initiation The workflow begins when a programming agent receives a `TASK.REQUEST` from the PI. Rather than immediately executing code, the agent first enters a planning phase by sending a `PLANNING.REQUEST` to itself. This self-messaging pattern allows the agent to leverage its planning capabilities to analyze the task context and select appropriate action units.

Planning-Execution Cycle The core workflow follows a planning-execution pattern where the agent first determines what to do, then executes it. After sending a `PLANNING.REQUEST`, the agent receives its own `PLANNING.RESPONSE` containing the selected action unit and reasoning. Based on the action unit's properties, the agent then sends a `CODE_WRITING.REQUEST` either to itself (for standard actions) or to the Domain Expert (for actions requiring specialized biomedical knowledge). This dynamic targeting ensures that each task receives appropriate expertise.

Code Review Loop Quality assurance is built into the communication protocol through an iterative review process. After generating code, the agent sends a `CODE REVIEW REQUEST` to either the Code Reviewer (for technical review) or the Domain Expert (for domain-specific validation). The

reviewer evaluates the code and returns a `CODE REVIEW RESPONSE` containing an approval/rejection decision and constructive feedback. If rejected, the agent processes the feedback and sends a `CODE REVISION REQUEST` to address the identified issues. This loop continues until the code is approved or the maximum number of review rounds is exceeded, balancing code quality with computational efficiency.

B.6 Timeout and Error Handling

Robust timeout and error handling mechanisms ensure that GenoMAS remains responsive and can gracefully recover from unexpected situations. The system implements a multi-level approach to manage execution time and handle errors.

At the agent level, each agent monitors its own execution time against a configured maximum. When an agent detects that it has exceeded its time limit, it emits a `TIMEOUT` message that signals the need for immediate task termination. This self-monitoring approach distributes the timeout detection responsibility, making the system more resilient.

The environment provides a second layer of timeout protection by tracking global execution time. During message processing, the environment checks whether agents have exceeded their time limits before allowing them to process new messages. When a timeout is detected at either level, the environment clears the message queue to prevent further processing and returns the current task results, ensuring that partial progress is preserved.

The system includes fallback mechanisms for handling errors, such as defaulting to predefined action sequences when parsing fails. This ensures that temporary failures do not derail entire workflows.

C Task-Specific Guidelines and Action Unit Prompts

This section presents the guidelines and action unit prompts that encode domain expertise for GEO and TCGA data preprocessing tasks, slightly simplified for brevity.

C.1 GEO Data Preprocessing Guidelines

The GEO agent follows these guidelines to transform raw Gene Expression Omnibus data into analysis-ready formats:

```
## GEO Data Preprocessing Workflow

Your task is to preprocess GEO series matrix data by extracting and
linking clinical features with gene expression values.

### Step 1: Data Loading and Initial Analysis
- Load the series matrix file using GEOFparse
- Identify clinical features in phenotype data
- Extract gene expression data matrix

### Step 2: Clinical Feature Processing
- Extract patient/sample identifiers
- Parse clinical variables from phenotype descriptions
- Handle special characters and formatting inconsistencies
- Create structured clinical dataframe

### Step 3: Gene Data Processing
- Identify gene annotation format (symbols, IDs, probes)
- Map identifiers to standard gene symbols using:
  * Internal gene synonym database
  * Platform annotation data
  * Probe-to-gene mappings
- Handle multi-mapped probes appropriately

### Step 4: Data Integration
- Ensure sample ID consistency between clinical and gene data
- Perform log transformation if data appears non-normalized
- Handle missing values using appropriate strategies
- Create final integrated dataset

### Quality Control Considerations
```

```

- Verify gene symbol mapping coverage (aim for >80%)
- Check for batch effects in expression data
- Validate clinical feature extraction accuracy
- Document any data quality issues

### Termination Conditions
- Complete integration of clinical and gene data achieved
- Major data quality issues prevent meaningful analysis
- Required clinical features cannot be extracted

```

C.2 GEO Action Unit Examples

The following action units implement specific steps in the GEO preprocessing workflow:

C.2.1 Initial Data Loading

```

Load the GEO series matrix file and perform initial exploration:
1. Load the data from the provided file path
2. Display basic dataset information and dimensions
3. Show sample phenotype information
4. Examine the expression data structure
5. Report the number of samples and features

```

C.2.2 Gene Annotation

```

Annotate genes with standardized symbols:
1. Examine the current gene identifiers
2. Determine identifier type (e.g., symbols, Entrez IDs, probe IDs)
3. Apply appropriate mapping strategy for normalization
4. Handle unmapped identifiers appropriately
5. Report mapping success rate and coverage statistics

```

C.3 TCGA Data Preprocessing Guidelines

The TCGA agent processes The Cancer Genome Atlas data with focus on clinical feature engineering:

```

## TCGA Data Preprocessing Workflow

Your task is to engineer demographic and clinical features from TCGA
data for downstream analysis.

### Step 1: Data Loading
- Load clinical and expression data files
- Verify data integrity and format
- Check sample ID correspondence

### Step 2: Feature Identification
- Scan clinical data for demographic variables:
  * Age, gender, race, ethnicity
  * Disease stage, grade, subtype
  * Treatment history
  * Survival outcomes
- Prioritize features with >70% completeness

### Step 3: Feature Engineering
- Standardize demographic categories
- Create binary indicators for key conditions
- Engineer interaction features if relevant
- Handle missing values appropriately:
  * Numerical: median/mean imputation
  * Categorical: mode or "Unknown" category

### Step 4: Validation
- Ensure all features have appropriate data types
- Verify value ranges and distributions
- Check for data entry errors

```

```

- Create summary statistics

### Termination Conditions
- At least 3 meaningful demographic features extracted
- Critical features have excessive missing data (>50%)
- Data quality issues prevent reliable feature engineering

```

C.4 TCGA Action Unit Examples

C.4.1 Find Candidate Demographic Features

Identify potential demographic features in clinical data:

1. List all available clinical variables
2. Search for demographic and clinical keywords
3. Assess data completeness for each candidate feature
4. Examine value distributions for categorical variables
5. Recommend features with sufficient data quality

C.4.2 Feature Engineering and Validation

Engineer and validate selected demographic features:

1. Standardize values for consistency
2. Apply appropriate missing value strategies
3. Create binary encodings where needed
4. Generate clinically meaningful interaction terms
5. Validate feature distributions and quality
6. Save processed features for downstream analysis

C.5 Statistician Guidelines

The Statistician agent receives preprocessed data and performs regression analyses following these guidelines:

```

## Statistical Analysis Workflow

Perform regression analyses to identify gene-trait associations.

### Analysis Types (in order of preference):
1. **Unconditional One-step**: Direct trait-gene regression
2. **Conditional One-step**: Include demographic covariates
3. **Two-step**: Batch correction followed by analysis

### Statistical Considerations:
- Use appropriate multiple testing correction (FDR/Bonferroni)
- Check regression assumptions (normality, homoscedasticity)
- Report both effect sizes and significance levels
- Generate diagnostic plots for model validation

### Output Requirements:
- Coefficient table with gene rankings
- Statistical significance metrics
- Model diagnostic visualizations
- Summary of top associated genes

```

These guidelines and prompts encode domain expertise while maintaining sufficient flexibility for agents to adapt to diverse datasets and analysis requirements.

D Guided Planning Framework and Metaprompts

This section provides technical details about the guided planning framework that enables programming agents to adaptively select action units based on task context and execution history.

D.1 Planning Mechanism Architecture

The planning mechanism implements a self-messaging pattern where programming agents send planning request messages to themselves at each decision point. This design enables agents to leverage their full language modeling capabilities for strategic decision-making while maintaining clear separation between planning and execution phases.

The planning process integrates several key components. Agents perform task context analysis by examining the complete execution history, including code outputs, errors, and debugging attempts. Based on this analysis, they select the most appropriate next action from available units. When retraction is enabled, agents can choose to backtrack to previous steps if critical issues are identified. All planning decisions follow a strict JSON schema to ensure reliable parsing and system robustness.

D.2 Planning Request Metaprompt

The following metaprompt guides programming agents through the planning process. The prompt adapts dynamically based on whether retraction is available:

```
You are {agent_role}. Your task is to execute a multi-step workflow.

# The following section appears only when retraction is available:
You have the option to retract to a previous step if you discover
critical issues that require revisiting earlier decisions.
Retraction remaining: {retract_remaining}

### High-level Guidelines:
{guidelines}

### Task History:
You have attempted the following actions:
{task_history}

### Next Action Selection:
Based on the task history, select the next action from:
{available_actions}

### Response Format:
Respond in JSON with the following structure:
{
    "action": "Selected Action Unit name",
    "retract_to_action": "Action to retract to (if applicable)",
    "reason": "Brief explanation of your decision"
}

Important considerations:
- Analyze all outputs and errors to understand the current state
- If critical errors prevent progress, consider alternative approaches
- Use retraction when fundamental issues require earlier corrections
- Select "TASK COMPLETED" only when all objectives are achieved
```

The metaprompt structure ensures that agents receive comprehensive context while maintaining focus on the immediate planning decision. The conditional inclusion of retraction instructions prevents confusion when this feature is disabled.

D.3 Planning Response Processing

The system implements robust parsing for planning responses with multiple fallback mechanisms. It first attempts direct JSON parsing of the model response. If this fails, it extracts JSON from markdown code blocks. The system then employs fuzzy action matching using TF-IDF similarity to handle minor variations in action names. As a final fallback, when planning is disabled or parsing fails completely, the system follows predefined action sequences. This multi-layered approach ensures system robustness while maintaining the benefits of adaptive planning.

D.4 Context Management for Planning

The planning mechanism employs sophisticated context management to balance informativeness with token efficiency:

Task History Formatting The system maintains detailed records of each step, including action unit names with timestamps, generated code (truncated if exceeding length limits), execution outputs with error traces, and clear distinctions between regular steps and debugging attempts. This comprehensive history enables agents to make informed planning decisions based on complete task context.

Context Modes Different planning scenarios use tailored context presentations. The `all` mode provides complete history for comprehensive planning decisions, while `past` mode excludes the current attempt for revision scenarios. The `last` mode focuses only on the most recent attempt for targeted debugging efforts.

D.5 Retraction Mechanism Design

The retraction mechanism enables agents to recover from decisions that lead to downstream complications. When an agent decides to retract, it reverts both its task context and execution state to a previous action unit while preserving the complete task history. This allows the agent to learn from failed attempts while exploring alternative approaches. The mechanism tracks retraction usage through a configurable budget, preventing infinite loops while providing sufficient flexibility for complex problem-solving. This design ensures execution state consistency while preserving valuable task history for future planning decisions.

E Code Generation, Review, and Domain Consultation Mechanisms

This section provides detailed technical specifications of the code generation, review, and domain expert consultation mechanisms that form the core of GenoMAS’s multi-turn programming workflow.

E.1 Code Generation and Review Process

E.1.1 Code Generation Architecture

The code generation process in GenoMAS implements a context-aware approach that enables programming agents to produce code incrementally while maintaining coherence across multiple action units. Programming agents receive comprehensive task history that includes all previously executed code snippets with their outputs, error traces from failed attempts, and reviews from previous debugging iterations. This accumulated context enables agents to understand data structures created in earlier steps and avoid repeating errors.

Each code generation request focuses on a single action unit while maintaining awareness of the broader workflow. Agents generate code that can be concatenated with previous steps and executed sequentially. The system dynamically routes code generation requests based on action unit properties—standard programming tasks are handled by the programming agents themselves, while actions requiring biomedical expertise are routed to the Domain Expert agent.

Code Writing Metaprompt Programming agents receive the following structured prompt when generating code for each action unit:

```
### General Guidelines:  
{guidelines}  
  
### Function Tools:  
{tools_code}  
  
### Programming Setups:  
{path_setup}  
  
### Task History:  
# Complete execution history of previous steps
```

```

{formatted_task_history}

### TO DO: Programming
Now that you've been familiar with the task setups and current
status, please write the code following the instructions:

{action_unit_instruction}

NOTE:
ONLY IMPLEMENT CODE FOR THE CURRENT STEP. MAKE SURE THE CODE CAN BE
CONCATENATED WITH THE CODE FROM PREVIOUS STEPS AND CORRECTLY EXECUTED.

FORMAT:
```python
[your_code]
```

NO text outside the code block.
Your code:

```

This metaprompt structure ensures that agents have access to all necessary context while maintaining focus on the current action unit. The explicit formatting requirements prevent extraneous text that could interfere with code execution.

E.1.2 Code Review Process

The code review mechanism implements a sophisticated evaluation framework that balances functionality verification with instruction conformance. Each review receives the task history and current code attempt but explicitly excludes reviews from previous attempts at the same step. This isolation prevents cascading biases where an incorrect initial review could influence subsequent assessments.

Code review follows two primary criteria: functionality assessment (whether the code executes successfully and produces expected outputs) and conformance checking (whether the code follows instructions and achieves intended objectives). For domain-specific tasks, reviewers also assess whether biomedical inferences are reasonable.

Code Review Metaprompt The following prompt guides reviewers through systematic evaluation:

```

### Task History:
# Previous steps excluding current attempt's reviews
{past_context}

### TO DO: Code Review
The following code is the latest attempt for the current step and
requires your review. If previous attempts have been included in the
task history above, their presence does not indicate they succeeded
or failed, though you can refer to their execution outputs for context.
Only review the latest code attempt provided below.

# Current attempt details
{current_attempt}

Please review the code according to the following criteria:
1. *Functionality*: Can the code be successfully executed in the
   current setting?
2. *Conformance*: Does the code conform to the given instructions?
   # Domain trigger added for biomedical tasks

Provide suggestions for revision and improvement if necessary.

*NOTE*:
1. Your review is not concerned with engineering code quality. The
   code is a quick demo for a research project, so the standards
   should not be strict.
2. If you provide suggestions, please limit them to 1 to 3 key
   suggestions. Focus on the most important aspects, such as how to
   solve the execution errors or make the code conform to the
   instructions.

State your decision exactly in the format: "Final Decision: Approved"

```

```
or "Final Decision: Rejected."
```

E.1.3 Code Revision Process

When code is rejected during review, the revision mechanism enables iterative improvement. Programming agents receive comprehensive context including all previous attempts and their reviews, allowing them to learn from the full debugging history. The revision process explicitly acknowledges that reviewer feedback may occasionally be incorrect or impractical, instructing agents to implement suggestions selectively based on their assessment.

Code Revision Metaprompt The revision prompt provides structured guidance:

```
### Task History:  
# All previous attempts and their reviews  
{past_context_with_reviews}
```

The following code is the latest attempt for the current step and requires correction. If previous attempts have been included in the task history above, their presence does not indicate they succeeded or failed, though you can refer to their execution outputs for context. Only correct the latest code attempt provided below.

```
# Current attempt details  
{current_attempt}
```

Use the reviewer's feedback to help debug and identify logical errors in the code. While the feedback is generally reliable, it might occasionally include errors or suggest changes that are impractical in the current context. Make revisions where you agree with the feedback, but retain the original code where you do not.

```
Reviewer's feedback:  
{reviewer_feedback}
```

```
# Standard code formatting requirements  
{CODE_INDUCER}
```

E.2 Domain Expert Consultation

The domain expert consultation mechanism provides specialized biomedical knowledge for action units that require scientific interpretation beyond standard programming expertise. Unlike the standard code review process where the Code Reviewer evaluates already-written code, the Domain Expert is consulted *during* code generation for specific action units. This fundamental difference ensures that domain knowledge is incorporated from the outset rather than retroactively.

When consulting the Domain Expert, the system provides a modified context that emphasizes data characteristics over implementation details. This includes dataset metadata, summary statistics, and specific domain questions, with minimal technical programming history. Furthermore, domain expert consultations maintain continuity—when domain-generated code encounters errors, the same Domain Expert receives the error information for debugging, preserving the domain-specific reasoning thread throughout the process. This contrasts with standard code review where different agents may handle successive iterations.

E.3 Context Management and Quality Assurance

E.3.1 Context Management Strategies

The system implements sophisticated context management to optimize information flow while managing token limits. The TaskContext maintains a hierarchical structure that distinguishes regular steps from debugging attempts, with each step recording its type, index, action name, instruction, code, output, and error traces.

Different context presentations serve specific purposes: comprehensive history for planning decisions, filtered views for unbiased code review, and domain-focused formatting for biomedical reasoning tasks. For long workflows, the system implements dynamic pruning strategies—code snippets may be truncated with clear indicators, while error traces are preserved in full to ensure debugging effectiveness.

E.3.2 Quality Assurance Mechanisms

Several mechanisms ensure code quality throughout the generation process. The system allows configurable rounds of review and revision (typically 1-3) before accepting code, catching both execution errors and logical flaws while preventing infinite debugging loops. Successfully reviewed code can be cached as snippets for similar future action units, accelerating development while maintaining quality. All generated code undergoes immediate execution validation, catching syntax errors, runtime exceptions, and logical errors that produce incorrect outputs.

These comprehensive mechanisms work together to produce high-quality code that correctly implements complex multi-step workflows while adapting to diverse dataset characteristics and analysis requirements.

F Examples of Autonomous Behaviors by Agents

This appendix provides concrete examples of autonomous agent behaviors observed during GenoMAS execution, demonstrating how programming agents adaptively handle edge cases and errors beyond their prescribed instructions. These examples illustrate the system's robustness in processing diverse genomic datasets.

F.1 Example 1: Autonomous Error Correction in Clinical Trait Extraction (GSE98578)

In processing the Acute Myeloid Leukemia dataset GSE98578, the GEO agent encountered persistent failures in clinical trait extraction despite multiple revision attempts. Rather than continuing failed approaches, the agent autonomously decided to re-implement the entire clinical data extraction logic.

F.1.1 Context and Initial Failure

The agent initially attempted to extract clinical features using the standard library functions:

```
# Initial attempt (Step 2) - Failed due to undefined function
selected_clinical_df = geo_select_clinical_features(
    clinical_df=clinical_data,
    trait=trait,
    trait_row=trait_row,
    convert_trait=convert_trait,
    age_row=age_row,
    convert_age=convert_age,
    gender_row=gender_row,
    convert_gender=convert_gender
)
```

This failed because the `convert_trait` function was defined but not properly accessible in the execution context. Multiple debugging attempts failed to resolve the scoping issue.

F.1.2 Autonomous Recovery Strategy

In Step 7, the agent autonomously decided to override the previous implementation entirely:

```
# Autonomous re-implementation in Step 7
# Agent's comment: "Need to recreate the clinical data extraction
# since it wasn't successfully executed in Step 2"

def convert_trait(value):
    """Convert AML subtype to binary format.
    AMKL = 1, non-AMKL = 0
    """
    if value is None:
```

```

    return None

    if ':' in value:
        value = value.split(':', 1)[1].strip()

    if value.lower() == 'amkl':
        return 1
    elif value.lower() == 'non-amkl':
        return 0
    else:
        return None

# Re-execute the entire clinical data extraction
selected_clinical_data = geo_select_clinical_features(
    clinical_df=clinical_data,
    trait=trait,
    trait_row=trait_row,
    convert_trait=convert_trait,
    age_row=age_row,
    convert_age=None,
    gender_row=gender_row,
    convert_gender=None
)

```

This autonomous decision to completely re-implement rather than patch the existing code demonstrates the agent's ability to recognize when incremental fixes are insufficient and a fresh approach is needed.

F.2 Example 2: Retroactive Clinical Rule Application (GSE123088)

When processing the Obesity dataset GSE123088, the agent implemented clinical trait extraction logic correctly but initially forgot to apply it to the raw clinical data. The agent later recognized this omission and autonomously corrected it.

F.2.1 Initial Oversight

In Step 2, the agent defined comprehensive conversion functions but failed to execute the clinical feature extraction:

```

# Step 2: Defined functions but skipped execution
def convert_trait(value):
    """Convert trait value to binary (0=No Obesity, 1=Obesity)"""
    if pd.isna(value):
        return None
    if ':' in value:
        value = value.split(':', 1)[1].strip()
    if value == 'OBESITY' or 'OBESITY' in value:
        return 1
    elif value == 'HEALTHY_CONTROL' or value == 'Control':
        return 0
    else:
        return None

# Agent's reasoning: "Skip this step since we don't have the
# actual clinical data file"
print("Clinical data extraction would require the actual clinical data file.")

```

F.2.2 Autonomous Correction

Recognizing the oversight in Step 7, the agent autonomously applied the previously defined rules:

```

# Step 7: Autonomous correction
# Agent's comment: "Clinical Data Extraction (which was skipped in Step 2)"

clinical_df = geo_select_clinical_features(
    clinical_data,
    trait=trait,
    trait_row=trait_row,

```

```

    convert_trait=convert_trait,
    age_row=age_row,
    convert_age=convert_age,
    gender_row=gender_row,
    convert_gender=convert_gender
)

print(f"Extracted clinical data with shape: {clinical_df.shape}")
print("Clinical data preview:")
print(preview_df(clinical_df))

# Save the clinical data
os.makedirs(os.path.dirname(out_clinical_data_file), exist_ok=True)
clinical_df.to_csv(out_clinical_data_file)
print(f"Clinical data saved to {out_clinical_data_file}")

```

This behavior shows the agent's ability to maintain awareness of incomplete tasks across multiple steps and autonomously correct oversights without explicit instructions.

F.3 Example 3: Early Task Termination Discovery (GSE125771)

In processing the Stroke dataset GSE125771, the planning mechanism initially failed to recognize that clinical trait data was unavailable. The agent discovered this limitation during execution and appropriately terminated the preprocessing task.

F.3.1 Planning Oversight

The initial planning assumed stroke trait data would be available in the clinical characteristics:

```

# Step 2: Initial analysis
# For trait (Stroke):
# This is a study on atherosclerotic plaques in carotid arteries
# All samples are from carotid plaques
# However, there's no explicit indication about which patients had stroke

trait_row = None # No explicit stroke data available

```

F.3.2 Autonomous Task Termination

Upon discovering the absence of trait data, the agent made the autonomous decision to properly document the limitation and terminate:

```

# Step 7: Autonomous termination decision
# Agent's reasoning: "Based on the exploration, we can see that
# clinical data doesn't have stroke information"

# Mark dataset as not having trait data available
is_trait_available = False
is_gene_available = True

# Initial validation with is_final=False since we don't have trait data
validate_and_save_cohort_info(
    is_final=False,
    cohort=cohort,
    info_path=json_path,
    is_gene_available=is_gene_available,
    is_trait_available=is_trait_available
)

print("Dataset processing complete. The dataset has gene expression " +
      "data but lacks stroke trait information.")

```

This demonstrates the agent's ability to recognize fundamental data limitations and make appropriate decisions about task continuation, preventing wasted computational effort on impossible analyses.

F.4 Key Insights from Autonomous Behaviors

These examples reveal several important patterns in agent autonomy:

1. **Strategic Re-implementation:** Agents can recognize when incremental debugging is insufficient and autonomously choose to re-implement functionality from scratch (Example 1).
2. **Cross-step Error Correction:** Agents maintain awareness of incomplete tasks across workflow steps and can retroactively apply corrections without explicit prompting (Example 2).
3. **Intelligent Task Termination:** Agents can identify fundamental data limitations and make appropriate decisions about workflow continuation, saving computational resources (Example 3).
4. **Context-aware Problem Solving:** All examples demonstrate agents' ability to reason about the broader workflow context when making autonomous decisions, rather than focusing narrowly on immediate instructions and error messages.

These autonomous behaviors emerge from the combination of comprehensive task history, flexible planning mechanisms, and the agents' underlying language model capabilities. They significantly enhance GenoMAS's robustness in handling the inherent variability and complexity of genomic data analysis.

G Examples of Notes Written by Agents

Throughout GenoMAS execution, agents generate structured notes that document their observations, challenges encountered, and potential issues discovered during data preprocessing. These self-reported notes serve multiple purposes: they enable efficient human oversight of system decisions, facilitate debugging of complex workflows, and provide insights into data quality issues that may affect downstream analyses. This section presents representative examples from actual experiment runs, organized by severity level.

G.1 Note Generation Mechanism

GenoMAS implements a systematic note-taking protocol where each agent logs observations during task execution. Notes are categorized into three severity levels:

- **INFO:** Routine observations about data characteristics, successful operations, and standard processing decisions
- **WARNING:** Potential issues that may affect analysis quality but do not prevent execution, such as small sample sizes or missing covariates
- **ERROR:** Critical failures that prevent dataset processing, including missing trait data or irrecoverable technical issues

Each note includes the dataset identifier, severity level, and contextual information that enables rapid assessment of preprocessing outcomes.

G.2 INFO-Level Notes: Documenting Standard Processing

INFO notes constitute the majority of agent observations, documenting successful preprocessing steps and dataset characteristics. These notes provide audit trails for quality assurance and enable researchers to understand processing decisions.

G.2.1 Dataset Characteristics

Agents systematically document processed dataset dimensions and properties:

```
GSE92538 (Bipolar disorder):
INFO: Dataset contains 161 samples after preprocessing with 11741 genes.

GSE46449 (Bipolar disorder):
INFO: Dataset contains 88 samples after preprocessing with 19845 genes.

GSE124283 (Gaucher Disease):
INFO: Successfully processed; gene symbols normalised where synonym
file present.
```

These dimension reports enable quick verification that preprocessing preserved sufficient samples and features for meaningful analysis.

G.2.2 Technical Processing Decisions

Agents document specific technical choices made during preprocessing:

GSE123086 (Crohn's Disease):

INFO: Dataset successfully processed with gene expression data from CD4+ T cells across multiple diseases including Crohn's disease. Gene identifiers kept as Entrez IDs.

GSE117261 (Multiple sclerosis):

INFO: Processed; gene symbols normalised with synonym dictionary.

GSE34788 (Heart rate):

INFO: Gene-level matrix rebuilt; clinical trait consistent with STEP-2; proper normalisation applied.

These notes reveal how agents adapt to different identifier systems (Entrez IDs vs. gene symbols) and apply appropriate normalization strategies based on data characteristics.

G.2.3 Special Dataset Contexts

Agents provide contextual information about dataset origins and study designs:

GSE77790 (Esophageal Cancer):

INFO: Dataset contains esophageal cancer cell lines (TE8, TE9) as cases and other cancer cell lines as controls. Highly imbalanced with only 2 esophageal cancer samples out of 32 total.

GSE100843 (Esophageal Cancer):

INFO: Dataset from Barrett's esophagus vitamin D supplementation study with pre/post treatment samples from both Barrett's segment and normal mucosa.

GSE131027 (Esophageal Cancer):

INFO: Dataset includes a wide range of cancer types to identify pathogenic germline variants. 'Esophageal cancer' is treated as the case group.

These contextual notes help researchers understand potential limitations, such as severe class imbalance or indirect disease proxies.

G.3 WARNING-Level Notes: Identifying Potential Issues

WARNING notes alert researchers to conditions that may compromise analysis quality without preventing execution. These notes guide interpretation of results and highlight datasets requiring careful consideration.

G.3.1 Sample Size Limitations

Agents flag datasets with limited statistical power:

GSE186963 (Crohn's Disease):

WARNING: Very small sample size after preprocessing.

GSE68950 (Mesothelioma):

WARNING: Small sample size may limit statistical power.

GSE107754 (Mesothelioma):

WARNING: Trait distribution is severely biased in this dataset.

G.3.2 Missing Covariate Data

Agents identify when important demographic or clinical covariates are unavailable:

GSE181339 (Hypertension):
WARNING: Trait data not available; cohort unusable for association analysis.

GSE173263 (Large B-cell Lymphoma):
WARNING: Trait data not available in this cohort.

GSE114022 (Large B-cell Lymphoma):
WARNING: Trait data missing - cohort unusable for association analysis.

G.3.3 Data Distribution Issues

Agents detect problematic trait distributions:

GSE27597 (Von Willebrand Disease):
WARNING: Von Willebrand Disease trait is severely biased - all samples have the same value (0) in this COPD/emphysema study.

These warnings enable researchers to exclude problematic datasets from meta-analyses or apply appropriate statistical corrections.

G.4 ERROR-Level Notes: Critical Processing Failures

ERROR notes document failures that prevent dataset integration into the analysis pipeline. These notes provide diagnostic information for troubleshooting and dataset exclusion decisions.

G.4.1 Gene Mapping Failures

Technical incompatibilities in gene annotation prevent successful preprocessing:

GSE93114 (Bipolar disorder):
ERROR: Gene symbol mapping failed because no suitable gene symbol column could be identified in the platform annotation.

GSE158237 (Obesity):
ERROR: No column containing gene symbols could be identified in the platform annotation; probe-to-gene mapping impossible.

GSE35661 (Heart rate):
ERROR: Gene mapping failed due to an inconsistency between the probe IDs in the expression matrix (ENST-based) and the annotation in the SOFT file (Affymetrix-based).

G.4.2 Missing Trait Information

Absence of phenotype data prevents case-control analyses:

GSE67311 (Bipolar disorder):
ERROR: Trait has only 0 distinct values

GSE65399 (Mitochondrial Disorders):
ERROR: Dataset lacks trait information for Mitochondrial_Disorders. Only gene expression data is available.

GSE203241 (Multiple sclerosis):
ERROR: Dataset lacks Multiple Sclerosis case/control trait information required for association study.

G.4.3 Data Structure Incompatibilities

Fundamental data organization issues prevent processing:

GSE271700 (Obesity):
ERROR: Gene mapping failed. The SOFT file does not contain the necessary probe-to-gene-symbol annotation table.

```
GSE186963 (Crohn's Disease):
ERROR: Preprocessing failed due to an unrecoverable mismatch in
sample IDs between clinical and genetic data files.
```

G.5 Leveraging Notes for System Improvement

The structured note system enables several workflows for continuous improvement:

Quality Assurance Workflow Researchers can filter notes by severity to prioritize review efforts. ERROR notes immediately identify datasets requiring exclusion or manual intervention. WARNING notes guide sensitivity analyses and robustness checks. INFO notes provide comprehensive audit trails for reproducibility.

Data Curation Insights Patterns in ERROR notes reveal common issues in public genomic databases. For instance, the frequency of gene mapping errors highlights the need for standardized annotation practices. Missing trait data errors suggest opportunities for enhanced metadata curation in repository submissions.

System Enhancement Opportunities Recurring error patterns inform development priorities. For example, the prevalence of probe-to-gene mapping failures led to enhanced synonym dictionaries and platform-specific mapping strategies in subsequent GenoMAS versions.

G.6 Summary

The agent-generated notes demonstrate GenoMAS's ability to provide transparent, interpretable pre-processing workflows. By systematically documenting observations across severity levels, the system enables efficient human oversight while maintaining full automation. These notes not only facilitate immediate quality control but also provide valuable insights for improving both the system and underlying data resources. The examples presented here, drawn from hundreds of preprocessing runs across diverse diseases and platforms, illustrate how structured self-reporting enhances the reliability and interpretability of automated genomic data analysis.