

GenoMAS: Guided Multi-Agent Programming for Scientific Discovery from Genomic Data

Haoyang Liu

University of Illinois
Urbana-Champaign
hl57@illinois.edu

Yijiang Li

University of California,
San Diego
yijiangli@ucsd.edu

Haohan Wang

University of Illinois
Urbana-Champaign
haohanw@illinois.edu

Abstract

Gene expression analysis holds the key to many biomedical discoveries, yet extracting insights from raw transcriptomic data remains challenging due to the complexity of multiple large, semi-structured files and the need for extensive domain expertise. Current approaches either rely on inflexible workflows that fail in edge cases or fully autonomous agents that lack the precision required for rigorous scientific inquiry. **GenoMAS** addresses this by presenting a team of LLM-based agents that integrates structured workflows with autonomous adaptability, orchestrating diverse specialized agents through typed message-passing protocols. At its core lies a guided-planning framework where programming agents decompose task guidelines into Action Units and dynamically choose to advance, revise, bypass, or backtrack, maintaining logical coherence while adapting to the idiosyncrasies of genomic data. On the GenoTEX benchmark, GenoMAS achieves 89.13% CSC for preprocessing and 60.48% F_1 for identifying trait-associated genes, surpassing prior art by 10.61% and 16.85%. It again outperforms strong agent baselines on ScienceAgentBench and DiscoveryBench, two independent bioinformatics benchmarks beyond transcriptomics. Its analyses yield gene-phenotype associations that are biologically plausible and corroborated by the literature. Our code is available at <https://github.com/Liu-Hy/GenoMAS>.

1 Introduction

Scientific research increasingly depends on complex computational analysis, yet such analysis remains labor-intensive, error-prone, and difficult to scale. Across genomics (Conesa et al., 2016; Stark et al., 2019), drug development (Vichentijevikj et al., 2026), and personalized medicine (Ginsburg and Phillips, 2018), the analytical backbone involves highly structured workflows that integrate raw data, apply statistical models, and yield inter-

pretable results under constraints of domain knowledge and methodological rigor. Recent advances in large language models (LLMs) have enabled agents capable of task decomposition (Luo et al., 2025; OpenAI et al., 2024), tool interaction (Qin et al., 2023; Zhu et al., 2025a), and code generation (Dong et al., 2024; He et al., 2025). These developments raise the prospect that LLM-based agents might contribute to scientific discovery by automating analysis pipelines (Alkan et al., 2025; Schmidgall et al., 2025). However, realizing this requires bridging the gap between general reasoning and the precision-driven nature of scientific computation.

While many LLM-based agents show competence in document retrieval or planning tasks (Zhuge et al., 2024; Yehudai et al., 2026), these capabilities fall short where scientific progress depends on code. In transcriptomics (Rosati et al., 2024), protein engineering (Swanson et al., 2024), and statistical genetics (Ginsburg and Phillips, 2018), workflows are encoded as programmatic transformations tailored to dataset idiosyncrasies. Analysts routinely write custom scripts to handle malformed metadata, adjust for confounders, map deprecated identifiers, and address batch effects (Johnson et al., 2007). These operations require direct, editable code and cannot be abstracted or offloaded to black-box tools. Even with standardized pipelines, deviations remain common. Scientific automation therefore hinges on writing, revising, and validating executable code.

Despite growing interest in agentic AI (Chen et al., 2025a), current systems rarely address this requirement. Most frameworks operate by composing tool calls (Qin et al., 2023) or coordinating invocations across structured graphs (Ye et al., 2025; Zhang et al., 2025b). Some support dynamic routing, but operate within fixed templates. A few attempt code-level control (Hu et al., 2025), but

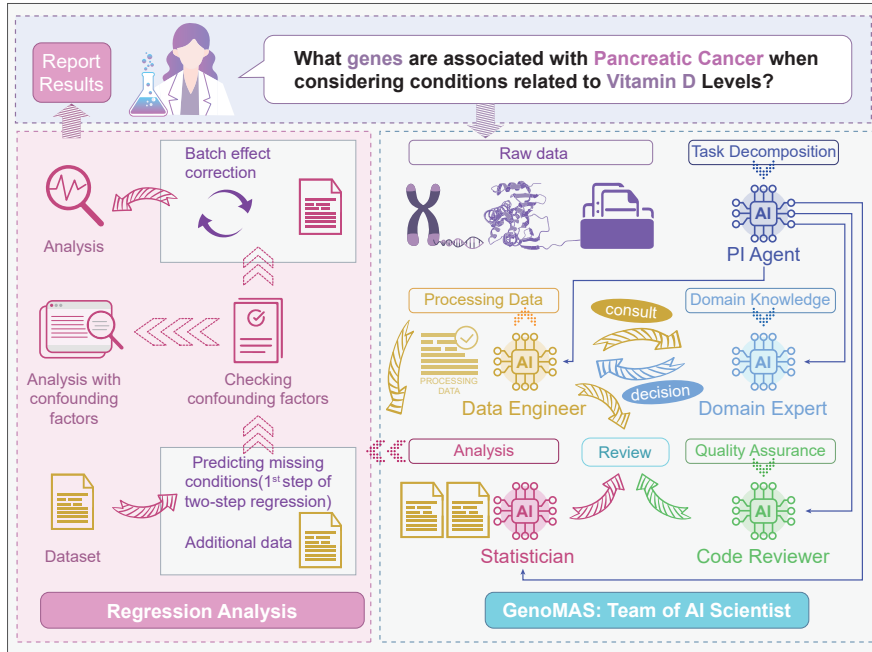


Figure 1: Multi-agent collaboration in our GenoMAS method. Given a user query, typically a conditional GTA question, the PI agent decomposes it into tasks; the Data Engineers preprocess the raw cohort data; and the Statistician performs the confounder-adjusted regression analysis. Throughout, the programming agents consult the Domain Expert for biology-dependent decisions and the Code Reviewer for code validation as needed, and the identified trait-associated genes are returned in an analysis report.

lack structured revision or domain-aware correction. Critically, these frameworks are benchmarked on synthetic tasks (Zhu et al., 2025a), where correctness is judged by output format rather than scientific validity. In gene expression analysis, even small preprocessing errors can invalidate results (Leek et al., 2010). Yet current systems offer no reliable means to detect and recover from such errors, remaining unsuitable for workflows where correctness is inseparable from domain context.

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 re-frames scientific agents as collaborative programmers rather than workflow orchestrators (Figure 1). 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 manual scripting by domain experts. Evaluated on GenoTEX (Liu et al., 2025a), a benchmark of 1,384 gene-trait association tasks spanning heterogeneous cohorts and confounders, GenoMAS

achieves state-of-the-art performance while operating from raw data inputs. The same design succeeds on ScienceAgentBench and DiscoveryBench, benchmarks spanning broader computational biology (Section 5.4). These results suggest that effective scientific automation requires agents that code, not merely reason and retrieve.

Our contributions are as follows:

1. We introduce **GenoMAS**, a multi-agent approach that treats agents as collaborative programmers rather than tool orchestrators, enabling end-to-end code generation for genomic analysis.
2. We develop a guided planning mechanism that encodes workflows as editable Action Units, structured directives transformable into and modifiable as executable code, balancing precise control with autonomous error handling.
3. We demonstrate that GenoMAS supports heterogeneous agent composition, letting distinct LLMs with complementary strengths operate in coordinated roles within one execution loop.

2 Related Work

LLM-based Multi-Agent Systems LLMs have enabled autonomous agents capable of complex reasoning and task execution (Wang et al., 2024a; OpenAI et al., 2024; Yehudai et al., 2026). These agents leverage structured reasoning approaches (Yao et al., 2023; Besta et al., 2024), self-reflection mechanisms (Xi et al., 2023; Madaan et al., 2023), and external tool integration (Qin et al., 2023; Qian et al., 2025). Multi-agent collaboration further enhances performance through specialized role-playing (Hong et al., 2024; Dong et al., 2024), goal decomposition (Wei et al., 2022; Ning et al., 2024), and feedback mechanisms (Huang et al., 2024; Yin et al., 2023). Recent work has explored failure modes (Cemri et al., 2025) and training frameworks (Ma et al., 2024a) for such systems.

LLM Agents for Scientific Discovery LLM agents are increasingly being developed to automate scientific discovery (Lu et al., 2024; Baek et al., 2025; Yehudai et al., 2026), including hypothesis generation (Alkan et al., 2025) and peer review (Zhu et al., 2025b). Systems such as AI Scientist (Lu et al., 2024) and Agent Laboratory (Schmidgall et al., 2025) demonstrate autonomous completion of research cycles. Domain-specific applications span mathematics (Romera-Paredes et al., 2024), physics (Ma et al., 2024b), chemistry (Bran et al., 2023), biology (Huang et al., 2025), and medicine (Liu et al., 2025b).

Positioning of Our Work While existing agentic systems for scientific discovery have demonstrated capabilities in hypothesis generation (Baek et al., 2025), experimental design (Jia et al., 2024), or literature-driven reasoning (Kang et al., 2023; Zhu et al., 2025b), they rarely address settings where agents must write and revise executable code under scientific constraints. GenoMAS targets this gap by treating scientific automation as a coding problem rather than a retrieval or coordination problem. Closest to our setting, a growing family of bioinformatics agents automates biological data analysis, including BioMaster (Su et al., 2026), CellAgent (Xiao et al., 2024), K-Dense Analyst (Li et al., 2025), TransAgent (Zhang et al., 2025a), Biomni (Huang et al., 2025), and GenoAgent (Liu et al., 2025a). These systems share the broad planner-executor-reviewer pattern with iterative repair of the current step. Our novelty does not rest on this pattern, but on two mechanisms validated in

Section 5: cross-family role specialization, which assigns complementary model families to the planning, programming, review, and domain reasoning roles, whereas these systems report a single agent or a shared backbone across agents (CellAgent adds a vision model for evaluation); and global planning with committed-step retraction, which restores both the task context and the execution state to a previous step when downstream evidence reveals a problematic earlier decision, going beyond current-step repair or replanning.

3 Preliminaries

3.1 Problem Formulation

We address gene-trait association (GTA) analysis, a key task in computational biology that identifies genes whose expression patterns correlate with phenotypic traits (Ghosh and Chinnaiyan, 2005; Wu et al., 2009). Following the GenoTEX benchmark (Liu et al., 2025a), we consider two problem settings:

- **Unconditional GTA:** Given gene expression data $X \in \mathbb{R}^{n \times p}$ and trait values $y \in \mathbb{R}^n$, identify genes $G^* \subseteq \{1, \dots, p\}$ significantly associated with the trait.
- **Conditional GTA:** Given additional condition $c \in \mathbb{R}^n$ (e.g., age, gender, or comorbidity), identify genes G_c^* associated with the trait while accounting for this condition (Kyalwazi et al., 2023), enabling more precise insights for personalized medicine (Wang et al., 2022).

The analysis pipeline consists of three stages: (1) dataset selection from repositories like the Gene Expression Omnibus (GEO) (Clough and Barrett, 2016) and The Cancer Genome Atlas (TCGA) (Tomczak et al., 2015), (2) data preprocessing to extract and normalize gene expression and clinical features, and (3) statistical analysis using regularized regression models (Wu et al., 2009).

3.2 Evaluation

We evaluate methods using the GenoTEX benchmark (Liu et al., 2025a), which assesses performance across three pipeline stages:

Dataset Selection 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 among multiple candidates.

Data Preprocessing Preprocessing involves extracting gene expression data from different platforms. Microarray probes require mapping to gene symbols (Khondoker, 2006), while RNA-seq reads demand alignment and quantification (Conesa et al., 2016). Gene identifiers must be normalized across evolving nomenclatures (Sayers et al., 2022; Wei et al., 2023), and clinical metadata extracted from heterogeneous formats (Çetin and Yildiz, 2022). Quality is evaluated using: (i) Attribute Jaccard (AJ) measuring feature overlap, (ii) Sample Jaccard (SJ) assessing sample overlap, and (iii) Composite Similarity Correlation (CSC), capturing both structural and numerical fidelity.

Statistical Analysis GTA analysis employs Lasso regression (Tibshirani, 1996) and its variants (Wu et al., 2009), requiring batch effect correction (Johnson et al., 2007), adjustment for population stratification (Yu et al., 2006), and imputation of missing values (Liew et al., 2011). Evaluation uses F_1 score, Precision, Recall, AUROC (Hanley and McNeil, 1982), and GSEA Enrichment Score (Subramanian et al., 2005).

3.3 Technical Challenges

Automated GTA analysis presents intertwined challenges (Angermueller et al., 2016): (i) *High-dimensional sparsity*: datasets typically comprise over 20,000 genes across <1,000 samples (Johnstone, 2001); (ii) *Platform heterogeneity*: microarrays and RNA-seq require distinct processing pipelines (Abusamra, 2013; Stark et al., 2019); (iii) *Evolving gene nomenclature*: continuous changes in gene names necessitate robust normalization (Sayers et al., 2022; Wei et al., 2023); (iv) *Heterogeneous metadata*: phenotypic information appears in varied formats, requiring domain expertise to extract from free-text or indirect descriptions (Chen et al., 2024); and (v) *Confounding factors*: batch effects (Leek et al., 2010), population stratification (Yu et al., 2006), and hidden covariates (Gagnon-Bartsch and Speed, 2012) can introduce spurious associations. These challenges require systems that combine computational pipelines with biological understanding.

4 Method: GenoMAS

4.1 Overview

The complexity of gene expression analysis demands both *precise procedural control* and *adaptive error handling*. Conventional workflow-based approaches offer structured execution but lack

the adaptability to manage edge cases, leading to suboptimal performance (Section 5.3); fully autonomous agents are more adaptable but sacrifice the precision essential for rigorous scientific analyses (Appendix A).

GenoMAS reconciles these competing requirements through a multi-agent framework in which programming agents execute user-defined guidelines while retaining autonomy to handle exceptions. The system coordinates a team of specialized agents that communicate through structured protocols and employ context-aware decision-making to address edge cases, retry failed operations, and integrate domain-specific knowledge.

4.2 Multi-Agent Architecture

Agent Roles and Responsibilities GenoMAS employs six specialized agents organized into three functional categories: (i) The **orchestration agent** (*PI agent*) coordinates the analysis workflow by dynamically assigning tasks based on requirements and task dependencies. (ii) The **programming agents** perform core computational tasks: two *Data Engineer agents* handle data preprocessing. The *GEO agent* focuses on GEO datasets while the *TCGA agent* manages TCGA data, each specialized in its data formats and preprocessing challenges. The *Statistician agent* conducts statistical analyses, employing regression models to identify trait-associated genes while accounting for confounding factors. (iii) The **advisory agents** support programming agents through complementary expertise: the *Code Reviewer agent* validates generated code for functionality and instruction conformance and suggests revisions, while the *Domain Expert agent* provides biomedical insights for decisions such as clinical feature extraction and gene identifier mapping.

Diverse LLMs as Agents' Backbone GenoMAS leverages heterogeneous LLM backbones to exploit complementary strengths (Hong and Page, 2004; Page, 2007; Wang et al., 2025). We power programming agents with Claude Sonnet 4 (Anthropic, 2025a), selected for its strong agentic coding abilities. OpenAI o3 (OpenAI, 2025b), known for robust reasoning, 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 (DeepMind, 2025b), a top-performing model on scientific benchmarks (Rein et al., 2023; Center for AI Safety et al., 2026), powers the Domain Expert agent with broad scientific

knowledge, particularly in biology.

Task Orchestration The **PI agent** orchestrates analysis workflows by adhering to the dependency structure of gene expression analysis. For each GTA task, it identifies preprocessed cohorts and schedules preprocessing only for missing data, maximizing reuse across tasks with overlapping traits. Leveraging the independence of cohort-specific transformations, the agent parallelizes execution to reduce runtime. Upon completion, the Statistician agent selects suitable cohorts for downstream inference. Throughout the pipeline, the PI agent issues tasks via structured messages, monitors completion signals, and enforces timeouts to prevent unbounded execution.

Message-Driven Task Progression GenoMAS employs a typed message-passing protocol governing task progression through structured request-response cycles. Messages carry sender role, message type (e.g., `CODE_REVIEW_RESPONSE`), content, and target roles, enabling precise coordination. Upon completing a code segment, programming agents send review requests to the Code Reviewer or Domain Expert based on requirements. Typed responses determine subsequent actions: approval advances execution, while rejection triggers revision. A centralized environment manages interactions via a message queue, enforcing topological constraints to avoid circular dependencies. This framework lets task flows emerge organically while ensuring coherent execution (Appendix C).

4.3 Programming Agents

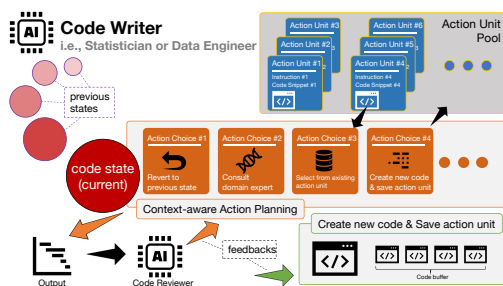


Figure 2: Planning, memory, and self-correction mechanisms of a single programming agent in our GenoMAS method.

4.3.1 Guided Planning

Guidelines and Action Unit The programming agents (**Data Engineers** and **Statistician**) execute complex workflows using a guided planning framework that balances structure and flexibility. Agents

operate under textual task guidelines encoding dependency structures, conditional logic, and termination criteria as a directed acyclic graph (DAG), enabling dynamic behavior adjustment based on evolving task context.

Within this DAG, agents decompose workflows into *Action Units*: semantically coherent operations corresponding 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 executable atomically, without intermediate oversight (guideline templates and Action Unit examples in Appendix D).

Guided Context-aware Planning At each step, programming agents analyze their task history and current state to select the next Action Unit, considering previous outcomes, discovered data characteristics, and remaining objectives. Agents can advance to the next Action Unit, revisit prior steps with modified parameters, skip optional steps, or terminate upon completion.

The retraction mechanism allows agents to backtrack when an earlier decision led to downstream complications. Upon retraction, agents revert task context and execution state to a previous step, then continue with an alternative Action Unit. This proves essential for handling complex interdependencies in gene expression analysis, where early preprocessing decisions can cascade through data quality. Appendix E provides additional details.

4.3.2 Domain Specific Code Generation

Iterative Code Generation and Debugging GenoMAS adopts a three-stage process (writing, review, revision) to generate robust pipelines. For each Action Unit, agents receive complete task context, including prior executions, error traces, and historical attempts, enabling reasoning about data structures and avoiding redundant mistakes. The Code Reviewer evaluates output, error messages, and guideline adherence, issuing approval or rejection. Agents then refine and resubmit code iteratively until approval or reaching debugging limits.

To ensure independent evaluation, code review enforces context isolation. The Code Reviewer receives the current attempt and task history but not prior review feedback at the same step, mitigating cascading biases. After review, programming agents regain access to all feedback and synthesize

it to revise the code.

Domain Expert Consultation for Biomedical Reasoning For Action Units requiring biomedical knowledge, programming agents consult the Domain Expert agent in place of the Code Reviewer. The expert 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 expert for debugging, enabling consistent reasoning across refinement rounds. Details and example prompts are provided in Appendix F.

Code Memory for Reuse Programming agents maintain a dynamic memory of validated code snippets indexed by Action Unit type; upon successful review, snippets are stored for reuse in similar contexts. This memory evolves with experience: agents can revise or replace stored code, improving efficiency and reliability while preserving flexibility to adapt to novel scenarios.

Specialized Tool Sets and Domain Knowledge Databases Each agent is equipped with specialized Python tools aligned with its responsibilities: Data Engineers access utilities for dataset loading, DataFrame manipulation, and gene mapping, while the Statistician employs statistical models and visualization libraries. Tools were co-developed by two computational biologists through iterative testing on genomic datasets, encapsulating best practices.

GenoMAS integrates local, version-controlled biological knowledge bases to ensure consistency. A curated gene synonym database from NCBI Gene (Brown et al., 2015) supports symbol normalization, while gene-trait association data from the Open Targets Platform (2025) enables prediction of missing conditions for conditional GTA tasks. The reference outputs used for evaluation are independent of Open Targets (see Section 6 for details).

5 Experiments

We evaluate **GenoMAS** on the GenoTEX benchmark (Liu et al., 2025a), which encompasses 1,384 GTA problems across 913 datasets (41.5 GB; 152,415 samples, 167 per cohort on average) spanning 132 human traits. GenoTEX covers the complete analytical workflow from raw data to biological insights, and provides expert-curated ground

truth for analyzing real-world genomic datasets with realistic complexities. These characteristics make it well-suited for assessing agentic methods on complex, multi-step analysis requiring scientific rigor.

Conditional GTA predominates: 1,252 problems (90.5%) are conditional and 132 (9.5%) unconditional, with conditions comprising age (130), gender (121), and another biomedical trait (1,001).

5.1 Experimental Setup

Computation environment We conducted experiments on 6 RunPod GPU Cloud (RunPod, 2024) instances, each with 16 vCPU cores and 94GB RAM. Running agent-generated code consumed about 0.3GB RAM across backbones. For LLM inference, we accessed proprietary models via their official APIs and open-source models through Novita AI (Novita AI, 2025) for lower latency.

Metrics Our end-to-end evaluation assesses the ability of GenoMAS to identify trait-associated genes from raw data. We adopt F_1 (Section 3.2) as the primary indicator, and report trait prediction accuracy and F_1 to validate the generated regression models. Runtime metrics include success rate (proportion of problems where generated code executes without errors and saves results to the correct path), token counts, API cost, and execution time per problem.

5.2 Comparison with Prior Art

We compare **GenoMAS** against the prior state-of-the-art method GenoAgent (Liu et al., 2025a), homogeneous variants where all agents share the same LLM backbone, and several baselines: random gene selection, and OpenAI o3 zero-shot, which attempts to directly identify trait-associated genes without access to data, probing whether parametric knowledge suffices. For comparison, we also include human expert performance reported from Liu et al. (2025a) – the average performance of two independent expert analyses when evaluated against a consensus ground truth.

Table 1 presents end-to-end results. GenoMAS achieves 60.48% F_1 score, yielding a 16.85% absolute improvement over GenoAgent, alongside 0.81 AUROC (0.17 increase) and 44.7% lower API cost. It attains 59.86% F_1 on conditional and 66.37% on unconditional GTA, and retains 84% of the F_1 of the fully human-conducted analysis (71.63%) without per-problem human intervention.

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% F_1 improvement and 48.9% cost reduction compared to homogeneous Claude-only configurations. Notably, Biomni, despite its success as a generalist biomedical agent, achieves only 14.82% F_1 even with the same task guidelines and Action Unit prompts as GenoMAS (Appendix A); workflow knowledge alone thus does not close the gap. Most other representative agents exhibited very low end-to-end success rates in preliminary experiments, so we exclude them here, except Claude Code, which we quantify below.

Comparison with a general-purpose coding agent To quantify the qualitative failures in Appendix A, we ran Claude Code (Anthropic, 2025), a widely used general-purpose coding agent, on all 1,384 problems under the protocol of Table 1, using one harness with two backbones: Claude Sonnet 4 (T.), matching GenoMAS’s programming backbone and thinking budget, and the newer Claude Sonnet 5 (Anthropic, 2026) with high-effort adaptive thinking (Appendix B.1). With the backbone fixed at Sonnet 4, Claude Code reaches 8.53% F_1 at a 12.14% success rate, versus 52.98% and 96.34% for homogeneous GenoMAS; as the two share backbone, guidelines, and metrics, this 44-point gap is attributable to the GenoMAS scaffold. Sonnet 5 lifts execution success to 99.42%, the highest in the table, yet F_1 reaches only 47.34%, below the GenoMAS scaffold on the older Sonnet 4 and far below full GenoMAS: backbone progress largely resolves execution completeness but does not yet close the gap in analysis accuracy our framework contributes. These gaps confirm that reliable gene expression analysis requires combining precise procedural control with agentic autonomy; Appendix A details the evaluation procedures and failure analyses.

5.3 Ablation Study

Ablation on each component We conducted systematic ablations of each component: (i) removing planning, forcing fixed-order execution of the static DAG; (ii) excluding the Domain Expert; (iii) restricting review to a single round; and (iv) omitting code review entirely.

Results in Table 1 demonstrate that context-aware planning enables dynamic adaptation to edge cases and error recovery, yielding both higher ac-

curacy and greater efficiency by eliminating redundant steps. The collaborative design, particularly the specialized Code Reviewer and Domain Expert agents, proves critical for scientific rigor, while multiple review rounds catch subtle errors with downstream impacts.

Individual Task Performance To identify performance bottlenecks, we evaluated the GenoMAS pipeline component-wise. For *dataset filtering and selection* (Figure 3(a)), agents exhibit reasonable effectiveness, likely due to the lower reasoning complexity of metadata-based relevance assessment; however, early-stage errors can cascade through the pipeline.

Data preprocessing (Figure 3(b)) reveals pronounced task-dependent variation. **GenoMAS** achieves high performance on gene expression data (91.15% CSC), demonstrating its effectiveness in handling genomic data transformation. In contrast, clinical trait preprocessing yields a markedly lower CSC of 32.61%, reflecting the heterogeneous nature of clinical data and the nuanced domain knowledge required for accurate extraction. This identifies clinical feature preprocessing as the main bottleneck, demanding both technical sophistication and biomedical expertise. Even so, this CSC remains the highest among all evaluated systems. A qualitative reading of GenoMAS trajectories (Appendix J) traces the residual errors to a few recurring judgment patterns, such as treating a within-patient contrast as a case-control label, and points to targeted safeguards as a promising direction for future work.

For *statistical analysis* (Table 2), we provided expert-preprocessed data as input, a controlled environment for testing GTA analysis accuracy. In this setting, GenoMAS with batch effect correction achieves 95.26% F_1 , substantially outperforming traditional regression baselines (Lasso: 14.03%), previous agent methods, and the variant without confound control.

A diagnostic tracked session shows the code memory mechanism substantially improves efficiency (Appendix G).

5.4 Generality Beyond Transcriptomics

While we focus on transcriptomic analysis, GenoMAS’s mechanisms are generic by design. We therefore evaluate it on two independently developed bioinformatics benchmarks, holding the code-generating model fixed across compared methods

Table 1: **End-to-end performance of GenoMAS, prior art, a general-purpose coding agent, homogeneous-LLM variants, ablations, human experts, and data-free baselines.** We report performance on GTA analysis, on the auxiliary trait prediction task, and on runtime 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 (Liu et al., 2025a)	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 (Huang et al., 2025)	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 Code (Sonnet 4 T.)	9.22	8.61	8.53	0.10	0.11	9.40	-	12.14	-	0.70	340.24
Claude Code (Sonnet 5 T.)	48.92	51.82	47.34	0.69	0.83	75.32	-	99.42	-	0.35	483.29
Claude Sonnet 4 (T.) (Anthropic, 2025a)	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 (Anthropic, 2025a)	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 (OpenAI, 2025b)	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 (OpenAI, 2025c)	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 (DeepMind, 2025b)	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 (DeepMind, 2025a)	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 (Guo et al., 2025)	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 (Yang et al., 2025)	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 (Liu et al., 2025a)	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 (Liu et al., 2025a)	0.72	0.90	0.75	0.49	0.04	-	-	-	-	-	0.02

Table 2: **Statistical analysis performance when using expert-preprocessed data as input.** Metrics are as in Table 1. Baselines include representative agents and LLMs, and fixed scripts implementing two standard regression models. “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 (Liu et al., 2025a)	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 (Huang et al., 2025)	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 (Hong et al., 2025)	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 (Hong et al., 2024)	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 (Wu et al., 2023)	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 (Wang et al., 2024b)	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 (OpenAI, 2025b)	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 (Guo et al., 2025)	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 (Tibshirani, 1996)	31.88	12.99	14.03	0.53	0.12	76.48	62.31	99.39	-	-	9.19
LMM (Henderson, 1950)	3.73	3.75	3.64	0.51	0.03	75.63	58.04	99.21	-	-	15.44

so that differences reflect agent architecture (protocols in Appendix B).

ScienceAgentBench Each of SAB’s 27 verified Bioinformatics tasks (Chen et al., 2025b) requires writing and executing a program that produces a specified scientific artifact, judged by the task’s own evaluator. GenoMAS (GPT-5 (OpenAI, 2025a) planning and review, Gemini 2.5 Pro expert, Claude Sonnet 4.5 (Anthropic, 2025b) coder) is compared with Direct and Self-debug controls sharing its coder, and with two public traces from the Holistic Agent Leaderboard (HAL) (Kapoor et al., 2026) re-scored under the same evaluator. GenoMAS attains the highest success

rate, 63.0% (17/27), against 44.4% (HAL SAB Self-Debug), 37.0% (Direct), 29.6% (Self-debug), and 25.9% (HAL generalist) (Wilson CIs in Table 3, Appendix B.2). Paired McNemar tests give significant gains over Direct ($p = .039$), Self-debug ($p = .004$), and the HAL generalist agent ($p = .002$); the 18.5-point gain over HAL SAB Self-Debug does not reach significance on 27 tasks ($p = .227$).

DiscoveryBench DiscoveryBench (Majumder et al., 2025) evaluates a structurally different form of discovery: analyzing a dataset and stating the resulting hypothesis in natural language, scored against the gold hypothesis by an LLM judge (Hy-

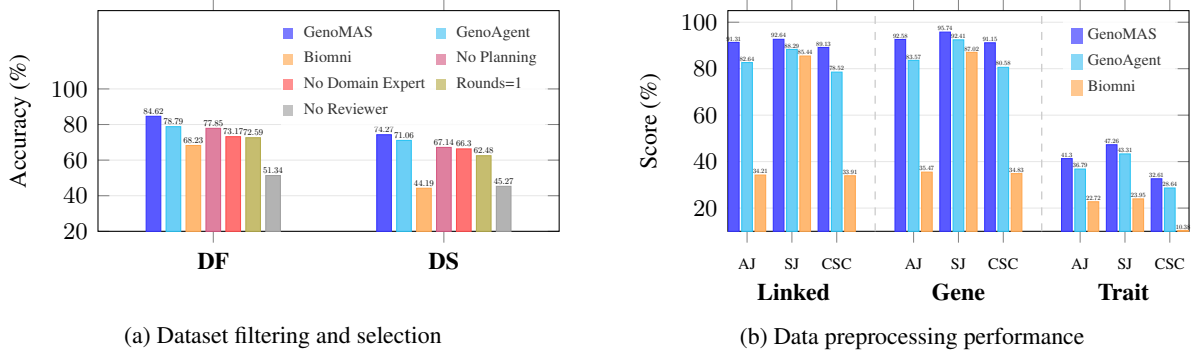


Figure 3: **Individual task performance on dataset filtering, selection, and preprocessing.** (a) Dataset filtering (DF) and selection (DS) accuracy of different methods and GenoMAS ablation settings. (b) Preprocessing performance across data types (Linked, Gene, Trait) and metrics (AJ: Attribute Jaccard; SJ: Sample Jaccard; CSC: Composite Similarity Correlation), with CSC primary.

pothesis Matching Score, HMS, 0 to 100); its 20 public biology tasks concern freshwater-fish evolution and non-native plant introduction pathways. Because nine tasks embed the answer in the query (a no-analysis response already scores about 70 HMS), we report the 11 discovery-requiring tasks defined by Direct’s failures (Table 4, Appendix B.3). GenoMAS reaches 29.4 HMS, against 13.6 for Self-debug and 8.1 for DiscoveryBench’s own coder agent on the same backbone. Removing guided planning drops GenoMAS to 11.1, into the range of the generic baselines, while removing code review is neutral (30.3): the margin here stems from guided planning, complementing the GenoTEX ablation, where review dominates because executable correctness is the bottleneck.

6 Qualitative Analysis

GenoMAS identifies biologically plausible gene-trait associations To assess whether GenoMAS produces scientifically interpretable results, we compared the genes it identified for unconditional GTA problems against known gene-disease associations from the Open Targets Platform (2025). The overlap yielded a GSEA enrichment score of 0.53 (FDR q -value 0.16), indicating moderate concordance with established disease biology. This comparison involves no circularity. GenoTEX’s reference outputs are curated by human bioinformaticians independently of Open Targets, and GenoMAS itself consults Open Targets only to estimate missing condition variables in conditional problems; no such queries arise for the unconditional problems examined here. Closer examination of high-confidence predictions for Alzheimer’s disease, lung cancer, and hypertension reveals

both well-established disease drivers (e.g., TERT for lung cancer) and biologically plausible candidates spanning disease-relevant mechanisms (Appendix H), supporting the use of GenoMAS outputs for hypothesis generation in genomic medicine.

Autonomous agent behaviors enhance workflow robustness Qualitative analysis of GenoMAS execution patterns reveals that agents autonomously adapt 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. In more complex cases, when an early decision creates downstream complications, agents can rewrite the code sequence from that decision point onward, effectively correcting the execution trajectory. These emergent behaviors, documented in Appendix I, show how agent autonomy strengthens robustness against the inherent variability of genomic data.

7 Conclusion

We presented GenoMAS, a multi-agent approach advancing automated gene expression analysis through three key contributions: guided planning that balances structured workflows with autonomous adaptation, heterogeneous LLM backbones with complementary strengths, and dynamic code memory for reuse. Evaluated on the GenoTEX benchmark, it reached state-of-the-art performance in GTA analysis at substantially lower cost, with gains that carry over to independent bioinformatics benchmarks. Future work will explore multi-modal data integration, richer cross-task memory, and more sophisticated planning algorithms.

Limitations

While GenoMAS demonstrates strong performance on gene expression analysis, several considerations merit discussion regarding its broader applicability and future development.

First, our framework inherits both the capabilities and limitations of its underlying LLMs. As these foundation models evolve, GenoMAS stands to benefit from improvements in reasoning depth and domain knowledge. However, this dependency also means that the system’s reliability is bounded by the current state of LLM development. Our heterogeneous agent architecture partially addresses this by strategically combining models with complementary strengths, yet systematic evaluation remains necessary as new model generations emerge.

Second, the trade-off between flexibility and efficiency represents a fundamental consideration for agentic scientific systems. While GenoMAS achieves notable cost reductions relative to prior agentic approaches, the iterative nature of agent-based planning and code generation introduces computational overhead compared to fixed pipelines. For scenarios prioritizing maximum throughput over adaptability, such as production-scale analyses with well-characterized data, practitioners may find value in distilling successful agent workflows into optimized static implementations.

Third, the GenoTEX results in Tables 1 and 2 aggregate problem-level outcomes from a single run per configuration over the 1,384 problems; with 26 configurations and 35,984 problem-level executions in total, repeated runs were computationally prohibitive, and we report paired significance tests only for the ScienceAgentBench comparisons (Section 5.4).

Finally, our primary evaluation focuses on regression-based gene-trait association analysis within human transcriptomics. The ScienceAgentBench and DiscoveryBench results indicate that GenoMAS’s mechanisms transfer to other computational-biology tasks, but neither benchmark provides a human-performance protocol comparable to GenoTEX’s, so a controlled multi-dataset human-AI comparison remains future work. Moreover, extending GenoMAS to encompass broader analytical paradigms (e.g., causal inference, network analysis) or integrating multi-omics modalities would require expanding both the agent capabilities and the domain knowledge bases. The modular architecture facilitates such

extensions, though each would necessitate domain-specific validation to ensure scientific rigor.

Ethical Considerations

The automation of scientific analysis through AI agents raises important considerations regarding research integrity, reproducibility, and responsible deployment.

Data privacy and security GenoMAS operates on publicly available genomic datasets from established repositories (GEO, TCGA) that have undergone institutional review and de-identification procedures. Our framework does not collect, store, or transmit private genetic information. However, practitioners applying GenoMAS to proprietary or sensitive datasets should ensure compliance with relevant data protection regulations (e.g., GDPR, HIPAA) and institutional ethical guidelines. The system’s reliance on external LLM APIs necessitates careful consideration of data transmission policies, particularly when cloud-based models process analysis code that may contain dataset identifiers or intermediate results.

Scientific rigor and interpretability While GenoMAS automates complex analytical workflows, the responsibility for validating biological conclusions remains with human researchers. Automated systems can efficiently process large-scale data and generate statistical associations, but interpreting these findings within broader biological contexts requires domain expertise and critical evaluation. We emphasize that GenoMAS serves as an analytical tool to assist, not replace, scientific judgment. The system’s self-documentation mechanisms (post-task notes, execution logs) are designed to facilitate expert oversight and enable researchers to understand and verify the analytical decisions made by agents.

Potential for misuse and dual-use concerns

The democratization of sophisticated bioinformatics capabilities through accessible AI systems offers substantial benefits for research accessibility and reproducibility. However, this accessibility also warrants consideration of potential misuse. While gene expression analysis itself represents standard computational biology practice, easier access to analytical tools could theoretically lower barriers for conducting research without appropriate biological training or ethical oversight. We advocate for responsible deployment within estab-

lished research frameworks, where automated analyses complement rather than circumvent traditional scientific training, peer review, and institutional oversight mechanisms.

Acknowledgements

We thank Ye Zhang, Senior Expert Data Scientist at Novartis, for his valuable suggestions. This research was supported by the National AI Research Resource (NAIRR) under grant number 240283, and by CloudBank through allocation CIS260773 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by U.S. National Science Foundation grants 2138259, 2138286, 2138307, 2137603, and 2138296.

References

- Heba Abusamra. 2013. [A comparative study of feature selection and classification methods for gene expression data of glioma](#). *Procedia Computer Science*, 23:5–14. 4th International Conference on Computational Systems-Biology and Bioinformatics, CS-Bio2013.
- Atilla Kaan Alkan, Shashwat Sourav, Maja Jablonska, Simone Astarita, Rishabh Chakrabarty, Nikhil Garuda, Pranav Khetarpal, Maciej Pióro, Dimitrios Tanoglidis, Kartheik G. Iyer, Mugdha S. Polimera, Michael J. Smith, Tirthankar Ghosal, Marc Huertas-Company, Sandor Kruk, Kevin Schawinski, and Ioana Ciucă. 2025. [A survey on hypothesis generation for scientific discovery in the era of large language models](#). *Preprint*, arXiv:2504.05496.
- Christof Angermueller, Tanel Pärnamaa, Leopold Parts, and Oliver Stegle. 2016. [Deep learning for computational biology](#). *Molecular Systems Biology*, 12(7):MSB156651.
- Anthropic. 2025. [Claude Code: Work with claude directly in your codebase. build, debug, and ship from your terminal, ide, slack, web, and more](#). Command line tool for agentic coding.
- Anthropic. 2025a. [Introducing Claude 4](#). Accessed: 2026-01-22.
- Anthropic. 2025b. [Introducing Claude Sonnet 4.5](#). Accessed: 2026-08-30.
- Anthropic. 2026. [Introducing Claude Sonnet 5](#). Accessed: 2026-08-30.
- Anysphere. 2024. [Cursor is your coding agent for building ambitious software](#). AI-powered code editor.
- Jinheon Baek, Sujay Kumar Jauhar, Silviu Cucerzan, and Sung Ju Hwang. 2025. [Researchagent: Iterative research idea generation over scientific literature with large language models](#). *Preprint*, arXiv:2404.07738.
- Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michał Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefer. 2024. [Graph of thoughts: solving elaborate problems with large language models](#). In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI’24/IAAI’24/EAAI’24. AAAI Press.
- Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. 2023. [Chemcrow: Augmenting large-language models with chemistry tools](#). *Preprint*, arXiv:2304.05376.
- Garth R Brown, Vichet Hem, Kenneth S Katz, Michael Ovetsky, Craig Wallin, Olga Ermolaeva, Igor Tolstoy, Tatiana Tatusova, Kim D Pruitt, Donna R Maglott, and Terence D Murphy. 2015. [Gene: a gene-centered information resource at NCBI](#). *Nucleic acids research*, 43(Database issue):D36–42.
- Odile Carrette, Isabelle Demalte, Alexander Scherl, Oezkarn Yalkinoglu, Garry Corthals, Pierre Burkhard, Denis F. Hochstrasser, and Jean-Charles Sanchez. 2003. [A panel of cerebrospinal fluid potential biomarkers for the diagnosis of alzheimer’s disease](#). *PROTEOMICS*, 3(8):1486–1494.
- Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. [Why do multi-agent llm systems fail?](#) *Preprint*, arXiv:2503.13657.
- Center for AI Safety, Scale AI, and HLE Contributors Consortium. 2026. [A benchmark of expert-level academic questions to assess AI capabilities](#). *Nature*, 649:1139–1146.
- Ke Chen, Peiran Wang, Yaoning Yu, Xianyang Zhan, and Haohan Wang. 2025a. [Large language model-based data science agent: A survey](#). *Preprint*, arXiv:2508.02744.
- Zhiwei Chen, Ziming Li, Xiaomin Niu, Xiangyun Ye, Yongfeng Yu, Shun Lu, and Zhen Chen. 2011. [The effect of CYP1A1 polymorphisms on the risk of lung cancer: a global meta-analysis based on 71 case-control studies](#). *Mutagenesis*, 26(3):437–446.
- Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. 2025b. [Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery](#). In *International Conference on Learning Representations*, volume 2025, pages 96934–96990.
- Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu,

- and Michael Cafarella. 2024. [Seed: Domain-specific data curation with large language models](#). *Preprint*, arXiv:2310.00749.
- Emily Clough and Tanya Barrett. 2016. [The gene expression omnibus database](#). *Methods in molecular biology (Clifton, N.J.)*, 1418:93 – 110.
- Ana Conesa, Pedro Madrigal, Sonia Tarazona, David Gomez-Cabrero, Alejandra Cervera, Andrew McPherson, Michał Wojciech Szcześniak, Daniel J. Gaffney, Laura L. Elo, Xuegong Zhang, and Ali Mortazavi. 2016. [A survey of best practices for rna-seq data analysis](#). *Genome Biology*, 17(1):13.
- Sascha David, Philipp Kümpers, Julian Hellpap, Rüdiger Horn, Holger Leitolf, Hermann Haller, and Jan T Kielstein. 2009. [Angiopietin 2 and cardiovascular disease in dialysis and kidney transplantation](#). *American Journal of Kidney Diseases*, 53(5):770–778.
- Google DeepMind. 2025a. [Gemini 2.5 flash](#). Fast performance thinking model for everyday tasks.
- Google DeepMind. 2025b. [Gemini 2.5: Our most intelligent ai model](#). Accessed: 2026-01-22.
- Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2024. [Self-collaboration code generation via chatgpt](#). *ACM Trans. Softw. Eng. Methodol.*, 33(7).
- Evangelos Evangelou, Helen R Warren, David Mosen-Ansorena, Borbala Mifsud, Raha Pazoki, He Gao, George Ntritsos, Niki Dimou, Claudia P Cabrera, Ibrahim Karaman, and 1 others. 2018. [Genetic analysis of over 1 million people identifies 535 new loci associated with blood pressure traits](#). *Nature Genetics*, 50(10):1412–1425.
- Johann A Gagnon-Bartsch and Terence P Speed. 2012. [Using control genes to correct for unwanted variation in microarray data](#). *Biostatistics*, 13(3):539–552.
- Debashis Ghosh and Arul M Chinnaiyan. 2005. [Classification and selection of biomarkers in genomic data using LASSO](#). *Journal of biomedicine & biotechnology*, 2005(2):147–154.
- Geoffrey S Ginsburg and Kathryn A Phillips. 2018. [Precision medicine: From science to value](#). *Health affairs (Project Hope)*, 37(5):694–701.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. [Deepseek-r1 incentivizes reasoning in llms through reinforcement learning](#). *Nature*, 645(8081):633–638.
- Qing Guo, Weiming Fu, Jun Xie, Hong Luo, Stephen F. Sells, J.W. Geddes, Vimala Bondada, Vivek M. Rangnekar, and Mark P. Mattson. 1998. [Par-4 is a mediator of neuronal degeneration associated with the pathogenesis of alzheimer disease](#). *Nature Medicine*, 4(8):957–962.
- J A Hanley and B J McNeil. 1982. [The meaning and use of the area under a receiver operating characteristic \(ROC\) curve](#). *Radiology*, 143(1):29–36.
- Junda He, Christoph Treude, and David Lo. 2025. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5).
- Charles Roy Henderson. 1950. Estimation of genetic parameters (abstract). *Annals of Mathematical Statistics*, 21:309–310.
- Lu Hong and Scott E. Page. 2004. [Groups of diverse problem solvers can outperform groups of high-ability problem solvers](#). *Proceedings of the National Academy of Sciences*, 101(46):16385–16389.
- Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Robert Tang, Xiangtao Lu, and 9 others. 2025. [Data interpreter: An LLM agent for data science](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19796–19821, Vienna, Austria. Association for Computational Linguistics.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiaowu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. [MetaGPT: Meta programming for a multi-agent collaborative framework](#). In *The Twelfth International Conference on Learning Representations*.
- Shengran Hu, Cong Lu, and Jeff Clune. 2025. [Automated design of agentic systems](#). *Preprint*, arXiv:2408.08435.
- Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. [Large language models cannot self-correct reasoning yet](#). *Preprint*, arXiv:2310.01798.
- Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Yusuf Roohani, Ryan Li, Lin Qiu, Gavin Li, Junze Zhang, Di Yin, Shruti Marwaha, Jennefer N. Carter, Xin Zhou, Matthew Wheeler, Jonathan A. Bernstein, Mengdi Wang, Peng He, Jingtian Zhou, and 4 others. 2025. [Biomni: A general-purpose biomedical ai agent](#). *bioRxiv*.
- Brett L Jennings, Seyhan Sahan-Firat, Anne M Estes, Kanak Das, Nasreen Farjana, Xiao R Fang, Frank J Gonzalez, and Kafait U Malik. 2010. [Cytochrome p450 1b1 contributes to angiotensin ii-induced hypertension and associated pathophysiology](#). *Hypertension*, 56(4):667–674.
- Shuyi Jia, Chao Zhang, and Victor Fung. 2024. [Llm4design: Autonomous materials discovery with large language models](#). *Preprint*, arXiv:2406.13163.

- W Evan Johnson, Cheng Li, and Ariel Rabinovic. 2007. [Adjusting batch effects in microarray expression data using empirical bayes methods](#). *Biostatistics*, 8(1):118–127.
- Iain M. Johnstone. 2001. [On the distribution of the largest eigenvalue in principal components analysis](#). *The Annals of Statistics*, 29(2):295 – 327.
- Hyeonsu B Kang, Nouran Soliman, Matt Latzke, Joseph Chee Chang, and Jonathan Bragg. 2023. [Committee: Literature discovery with personal elected author committees](#). In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA. Association for Computing Machinery.
- Sayash Kapoor, Benedikt Stroebel, Peter Kirgis, Nitya Nadgir, Zachary S Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, Franck Nd-zomga, Dheeraj Oruganty, Sophie Luskin, Kangheng Liu, Botao Yu, Amit Arora, Dongyoon Hahm, Harsh Trivedi, Huan Sun, and 12 others. 2026. [Holistic agent leaderboard: The missing infrastructure for AI agent evaluation](#). In *The Fourteenth International Conference on Learning Representations*.
- Levon M. Khachigian. 2006. [Early growth response-1 in cardiovascular pathobiology](#). *Circulation Research*, 98(2):186–191.
- Levon M Khachigian, Volkhard Lindner, Amy J Williams, and Tucker Collins. 1996. [Egr-1-induced endothelial gene expression: A common theme in vascular injury](#). *Science*, 271(5254):1427–1431.
- Md Mizanur Rahman Khondoker. 2006. *Statistical methods for preprocessing microarray gene expression data*. Ph.D. thesis, University of Edinburgh.
- Takashi Kohno, Hitoshi Ichikawa, Yasushi Totoki, Kazuki Yasuda, Masaki Hiramoto, Takao Nammo, Hiromi Sakamoto, Koji Tsuta, Koh Furuta, Yoko Shimada, Reika Iwakawa, Hideaki Ogiwara, Takahiro Oike, Masato Enari, Aaron J Schetter, Hirokazu Okayama, Aage Haugen, Vidar Skaug, Suenori Chiku, and 10 others. 2012. [Kif5b-ret fusions in lung adenocarcinoma](#). *Nature Medicine*, 18(3):375–377.
- Beverly Kyalwazi, Christina Yau, Michael J. Campbell, Toshio F. Yoshimatsu, A. Jo Chien, Anne M. Wallace, Andres Forero-Torres, Lajos Pusztai, Erin D. Ellis, Kathy S. Albain, Anne H. Blaes, Barbara B. Haley, Judy C. Boughey, Anthony D. Elias, Amy S. Clark, Claudine J. Isaacs, Rita Nanda, Hyo S. Han, Rachel L. Yung, and 15 others. 2023. [Race, gene expression signatures, and clinical outcomes of patients with high-risk early breast cancer](#). *JAMA Network Open*, 6(12):e2349646–e2349646.
- Jeffrey T. Leek, Robert B. Scharpf, Héctor Corrada Bravo, David Simcha, Benjamin Langmead, W. Evan Johnson, Donald Geman, Keith Baggerly, and Rafael A. Irizarry. 2010. [Tackling the widespread and critical impact of batch effects in high-throughput data](#). *Nature Reviews Genetics*, 11(10):733–739.
- Orion Li, Vinayak Agarwal, Summer Zhou, Ashwin Gopinath, and Timothy Kassis. 2025. [K-dense analyst: Towards fully automated scientific analysis](#). *Preprint*, arXiv:2508.07043.
- Alan Wee-Chung Liew, Ngai-Fong Law, and Hong Yan. 2011. [Missing value imputation for gene expression data: computational techniques to recover missing data from available information](#). *Briefings in Bioinformatics*, 12(5):498–513.
- Doron Lipson, Marzia Capelletti, Roman Yelensky, Geoff Otto, Alex Parker, Mirna Jarosz, John A Curran, Sohail Balasubramanian, Troy Bloom, Kristina W Brennan, Amy Donahue, Sean R Downing, Garrett M Frampton, Lazaro Garcia, Frank Juhn, Kathy C Mitchell, Emily White, Jared White, Zac Zwirko, and 13 others. 2012. [Identification of new alk and ret gene fusions from colorectal and lung cancer biopsies](#). *Nature Medicine*, 18(3):382–384.
- Haoyang Liu, Shuyu Chen, Ye Zhang, and Haohan Wang. 2025a. [Genotex: An llm agent benchmark for automated gene expression data analysis](#). *Preprint*, arXiv:2406.15341.
- Sizhe Liu, Yizhou Lu, Siyu Chen, Xiyang Hu, Jieyu Zhao, Yingzhou Lu, and Yue Zhao. 2025b. [Drugagent: Automating ai-aided drug discovery programming through llm multi-agent collaboration](#). *Preprint*, arXiv:2411.15692.
- Gervaise Loirand, Patrice Guérin, and Pierre Pacaud. 2006. [Rho kinases in cardiovascular physiology and pathophysiology](#). *Circulation Research*, 98(3):322–334.
- Gervaise Loirand and Pierre Pacaud. 2010. [The role of rho protein signaling in hypertension](#). *Nature Reviews Cardiology*, 7(11):637–647.
- Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. [The ai scientist: Towards fully automated open-ended scientific discovery](#). *Preprint*, arXiv:2408.06292.
- Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, Rongcheng Tu, Xiao Luo, Wei Ju, Zhiping Xiao, Yifan Wang, Meng Xiao, Chenwu Liu, Jingyang Yuan, Shichang Zhang, and 7 others. 2025. [Large language model agent: A survey on methodology, applications and challenges](#). *Preprint*, arXiv:2503.21460.
- Hao Ma, Tianyi Hu, Zhiqiang Pu, Boyin Liu, Xiaolin Ai, Yanyan Liang, and Min Chen. 2024a. [Coevolving with the other you: Fine-tuning llm with sequential cooperative multi-agent reinforcement learning](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 15497–15525. Curran Associates, Inc.
- Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B. Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. 2024b. [LLM](#)

- and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 33940–33962. PMLR.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. *Self-refine: Iterative refinement with self-feedback*. *Preprint*, arXiv:2303.17651.
- Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. 2025. *Discoverybench: Towards data-driven discovery with large language models*. In *International Conference on Learning Representations*, volume 2025, pages 4556–4579.
- Kafait U Malik, Brett L Jennings, Fariborz A Yaghini, Seyhan Sahan-Firat, Chi Young Song, Anne M Estes, and Xiao R Fang. 2012. *Contribution of cytochrome p450 1b1 to hypertension and associated pathophysiology: a novel target for antihypertensive agents*. *Prostaglandins & Other Lipid Mediators*, 98(3–4):69–74.
- James D. McKay, Rayjean J. Hung, Valerie Gaborieau, Paolo Boffetta, Amelie Chabrier, Graham Byrnes, David Zaridze, Anush Mukeria, Neonilia Szeszenia-Dabrowska, Jolanta Lissowska, Peter Rudnai, Eleonora Fabianova, Dana Mates, Vladimir Bencko, Lenka Foretova, Vladimir Janout, John McLaughlin, Frances Shepherd, Alexandre Montpetit, and 55 others. 2008. *Lung cancer susceptibility locus at 5p15.33*. *Nature Genetics*, 40(12):1404–1406.
- Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. 2024. *Skeleton-of-thought: Prompting LLMs for efficient parallel generation*. In *The Twelfth International Conference on Learning Representations*.
- Novita AI. 2025. The ai-native cloud for builders and agents. <https://novita.ai>. Accessed: 2025-02-17.
- Open Targets. 2025. *Open targets platform*. Accessed: February 24, 2025.
- OpenAI. 2025a. *Introducing GPT-5*. Accessed: 2026-08-30.
- OpenAI. 2025b. *Introducing openai o3 and o4-mini*. Accessed: 2026-01-22.
- OpenAI. 2025c. *Openai o3-mini*. Accessed: 2026-01-22.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. *Gpt-4 technical report*. *Preprint*, arXiv:2303.08774.
- S.E. Page. 2007. *The Difference: How the Power of Diversity Creates Better Groups, Firms, Schools, and Societies*. The William G. Bowen Memorial Series in Higher Education Series. Princeton University Press.
- Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiushi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. *Toolrl: Reward is all tool learning needs*. *Preprint*, arXiv:2504.13958.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. *Toolllm: Facilitating large language models to master 16000+ real-world apis*. *Preprint*, arXiv:2307.16789.
- Thorunn Rafnar, Patrick Sulem, Simon N Stacey, Frank Geller, Julius Gudmundsson, Asgeir Sigurdsson, Margret Jakobsdottir, Hafdis Helgadóttir, Steinunn Thorlacius, Katja K H Aben, Thorarinn Blöndal, Thorgeir E Thorgeirsson, Gudmar Thorleifsson, Kristleifur Kristjansson, Kristin Thorisdottir, Rafn Ragnarsson, Bardur Sigurgeirsson, Halla Skuladottir, Tomas Gudbjartsson, and 66 others. 2009. *Sequence variants at the tert-clptm11 locus associate with many cancer types*. *Nature genetics*, 41(2):221–227.
- Stefanie Rauskolb, Thomas Andreska, Sophie Fries, Cora Ruedt von Collenberg, Robert Blum, Camelia-Maria Monoranu, Carmen Villmann, and Michael Sendtner. 2022. *Insulin-like growth factor 5 associates with human Aβ plaques and promotes cognitive impairment*. *Acta Neuropathologica Communications*, 10(1).
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. *GPQA: A graduate-level google-proof Q&A benchmark*. *Preprint*, arXiv:2311.12022.
- Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, and 1 others. 2024. *Mathematical discoveries from program search with large language models*. *Nature*, 625(7995):468–475.
- Diletta Rosati, Maria Palmieri, Giulia Brunelli, Andrea Morrione, Francesco Iannelli, Elisa Frullanti, and Antonio Giordano. 2024. *Differential gene expression analysis pipelines and bioinformatic tools for the identification of specific biomarkers: A review*.

- Computational and Structural Biotechnology Journal*, 23:1154–1168.
- RunPod. 2024. The ai developer cloud. <https://www.runpod.io/>. Accessed: 2024-06-06.
- Arnold M. Salazar, Amanda M. Leisgang, Andrew A. Ortiz, Andrew S. Murtishaw, and Jefferson W. Kinney. 2021. [Alterations of gaba b receptors in the app/ps1 mouse model of alzheimer’s disease](#). *Neurobiology of Aging*, 97:129–143.
- Eric W Sayers, Evan E Bolton, J Rodney Brister, Kathi Canese, Jessica Chan, Donald C Comeau, Ryan Connor, Kathryn Funk, Chris Kelly, Sunghwan Kim, and 1 others. 2022. [Database resources of the national center for biotechnology information](#). *Nucleic Acids Research*, 50(D1):D20–D26.
- Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. 2025. [Agent laboratory: Using llm agents as research assistants](#). *Preprint*, arXiv:2501.04227.
- Minjie Shen, Yanshu Li, Lulu Chen, Zhichao Fan, Yanhang Li, Qikai Yang, and Haochen Yang. 2026. [From mind to machine: The rise of manus ai as a fully autonomous digital agent](#). *Preprint*, arXiv:2505.02024.
- Xiuquan Shi, Suhua Zhou, Zhongxu Wang, Zongcan Zhou, and Zengzhen Wang. 2008. Cyp1a1 and gstm1 polymorphisms and lung cancer risk in chinese populations: a meta-analysis. *Lung Cancer*, 59(2):155–163.
- Rukshana C Shroff, Karen L Price, Maria Kolatsi-Joannou, Alexandra F Todd, David Wells, John Deanfield, Richard J Johnson, Lesley Rees, Adrian S Woolf, and David A Long. 2013. [Circulating angiopoietin-2 is a marker for early cardiovascular disease in children on chronic dialysis](#). *PLoS ONE*, 8(2):e56273.
- Rory Stark, Marta Grzelak, and James Hadfield. 2019. [Rna sequencing: the teenage years](#). *Nature Reviews Genetics*, 20(11):631–656.
- Houcheng Su, Junning Feng, Yawen Lu, Yucheng Xu, Jinming Yang, Haojie Lu, Jixin Yang, Xu Yang, Sirui Xie, Weicai Long, Chengrui Wang, Yusen Hou, Tingyu Zhu, and Yanlin Zhang. 2026. [Biomaster: Multi-agent system for automated bioinformatics analysis workflow](#). *Patterns*, 7(8):101611.
- Aravind Subramanian, Pablo Tamayo, Vamsi K. Mootha, Sayan Mukherjee, Benjamin L. Ebert, Michael A. Gillette, Amanda Paulovich, Scott L. Pomeroy, Todd R. Golub, Eric S. Lander, and Jill P. Mesirov. 2005. [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.
- Kyle Swanson, Wesley Wu, Nash L Bulaong, John E Pak, and James Zou. 2024. [The virtual lab: Ai agents design new sars-cov-2 nanobodies with experimental validation](#). *bioRxiv*, page 2024.11.11.623004.
- Robert Tibshirani. 1996. [Regression shrinkage and selection via the lasso](#). *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.
- Katarzyna Tomczak, Patrycja Czerwińska, and Maciej Wiznerowicz. 2015. [The cancer genome atlas \(TCGA\): an immeasurable source of knowledge](#). *Contemporary oncology (Poznan, Poland)*, 19(1A):A68–77.
- Ivana Vichentijevikj, Kostadin Mishev, and Monika Simjanoska Misheva. 2026. [Prompt-to-pill: Multi-agent drug discovery and clinical simulation pipeline](#). *Bioinformatics Advances*, 6(1):vbaf323.
- Haohan Wang, Bryon Aragam, and Eric P Xing. 2022. [Trade-offs of linear mixed models in genome-wide association studies](#). *Journal of computational biology : a journal of computational molecular cell biology*, 29(3):233–242.
- Junlin Wang, Jue WANG, Ben Athiwaratkun, Ce Zhang, and James Zou. 2025. [Mixture-of-agents enhances large language model capabilities](#). In *The Thirteenth International Conference on Learning Representations*.
- Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024a. [A survey on large language model based autonomous agents](#). *Frontiers of Computer Science*, 18(6).
- Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. [Executable code actions elicit better llm agents](#). *Preprint*, arXiv:2402.01030.
- Chih-Hsuan Wei, Ling Luo, Rezarta Islamaj, Po-Ting Lai, and Zhiyong Lu. 2023. [Gnorm2: an improved gene name recognition and normalization system](#). *Bioinformatics*, 39(10):btad599.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. [Chain-of-thought prompting elicits reasoning in large language models](#). In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. [Autogen: Enabling next-gen llm applications via multi-agent conversation](#). *Preprint*, arXiv:2308.08155.

- Tong Tong Wu, Yi Fang Chen, Trevor Hastie, Eric Sobel, and Kenneth Lange. 2009. [Genome-wide association analysis by lasso penalized logistic regression](#). *Bioinformatics*, 25(6):714–721.
- Zhiheng Xi, Senjie Jin, Yuhao Zhou, Rui Zheng, Songyang Gao, Jia Liu, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. [Self-Polish: Enhance reasoning in large language models via problem refinement](#). In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11383–11406, Singapore. Association for Computational Linguistics.
- Yihang Xiao, Jinyi Liu, Yan Zheng, Xiaohan Xie, Jianye Hao, Mingzhi Li, Ruitao Wang, Fei Ni, Yuxiao Li, Jintian Luo, Shaoqing Jiao, and Jiajie Peng. 2024. [Cellagent: An llm-driven multi-agent framework for automated single-cell data analysis](#). *Preprint*, arXiv:2407.09811.
- Jun Xie and Qing Guo. 2005. [Par-4 is involved in regulation of \$\beta\$ -secretase cleavage of the alzheimer amyloid precursor protein](#). *Journal of Biological Chemistry*, 280(14):13824–13832.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. [Qwen3 technical report](#). *Preprint*, arXiv:2505.09388.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. [Tree of Thoughts: Deliberate problem solving with large language models](#). *Preprint*, arXiv:2305.10601.
- Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. 2025. [Mas-gpt: Training llms to build llm-based multi-agent systems](#). *Preprint*, arXiv:2503.03686.
- Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2026. [Survey on evaluation of llm-based agents](#). *Preprint*, arXiv:2503.16416.
- Zhangyue Yin, Qiushi Sun, Cheng Chang, Qipeng Guo, Junqi Dai, Xuanjing Huang, and Xipeng Qiu. 2023. [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, Singapore. Association for Computational Linguistics.
- Jianming Yu, Gael Pressoir, William H. Briggs, Irie Vroh Bi, Masanori Yamasaki, John F. Doebley, Michael D. McMullen, Brandon S. Gaut, Dahlia M. Nielsen, James B. Holland, Stephen Kresovich, and Edward S. Buckler. 2006. [A unified mixed-model method for association mapping that accounts for multiple levels of relatedness](#). *Nature Genetics*, 38(2):203–208.
- Guorui Zhang, Chao Song, Liyuan Liu, Qiuyu Wang, and Chunquan Li. 2025a. [TransAgent: Dynamizing transcriptional regulation analysis via multi-omics-aware AI agent](#). *bioRxiv*. Doi: 10.1101/2025.04.27.650826.
- Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025b. [Aflow: Automating agentic workflow generation](#). *Preprint*, arXiv:2410.10762.
- Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. 2025a. [Multiagentbench: Evaluating the collaboration and competition of llm agents](#). *Preprint*, arXiv:2503.01935.
- Minjun Zhu, Yixuan Weng, Linyi Yang, and Yue Zhang. 2025b. [Deepreview: Improving llm-based paper review with human-like deep thinking process](#). *Preprint*, arXiv:2503.08569.
- Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. [GPTSwarm: Language agents as optimizable graphs](#). In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 62743–62767. PMLR.
- Volkan Çetin and Oktay Yildiz. 2022. [A comprehensive review on data preprocessing techniques in data analysis](#). *Pamukkale University Journal of Engineering Sciences*, 28:299–312.

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 documents the protocols of the benchmark-wide evaluation of a general-purpose coding agent and of the two external benchmark evaluations, along with the inference configuration.
- Appendix C describes the communication protocols and messaging mechanisms in GenoMAS.
- Appendix D presents the task-specific guidelines and action unit prompts for GEO and TCGA preprocessing.
- Appendix E details the guided planning framework and metaprompts for programming agents.
- Appendix F presents the code generation, review, and domain consultation mechanisms with their metaprompts.
- Appendix G analyzes the memory and code reuse efficiency in GenoMAS.
- Appendix H presents exploratory biological plausibility case studies for selected GenoMAS gene-trait associations.
- Appendix I presents some examples of agent autonomy in GenoMAS.
- Appendix J analyzes recurring failure cases in clinical feature extraction and outlines targeted safeguards.
- Appendix K presents some examples of notes written by agents.
- Appendix L visualizes the agent collaboration and communication patterns in GenoMAS.

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 min-

imal user input. To evaluate their applicability to gene expression analysis, we systematically tested six representative agents: three open-source systems (Biomni (Huang et al., 2025), AutoGen (Wu et al., 2023), MetaGPT (Hong et al., 2024)) and three proprietary systems (Claude Code (Anthropic, 2025), Cursor (Anysphere, 2024), Manus (Shen et al., 2026)). For all agents, we use Claude Sonnet 4 (Thinking) with a thinking budget of 8192 tokens as the underlying language model. This configuration represents the best performing setting in our end-to-end evaluation with homogeneous LLM configurations (achieving F_1 of 52.98%), and most of these agents do not support heterogeneous multi-model configurations. 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%).

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.

The remaining five agents, designed for open-domain or general programming tasks, did not produce complete, valid analyses in our preliminary

trials. We subsequently evaluated one of them, Claude Code, end to end on all 1,384 benchmark problems (Section 5.2; protocol in Appendix B.1). Its 12.14% execution success rate on the Claude Sonnet 4 backbone confirms that the failure modes illustrated below are not confined to the selected cohorts.

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 remainder of this section presents examples and summary observations 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 (Liu et al., 2025a) 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 D). For readability, code blocks below are excerpts from the audited outputs and are not complete execution logs.

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 trait assignment is misaligned with the benchmark phenotype definition used for this cohort. Liver cirrhosis accounts for approximately 90% of portal hypertension cases in Western countries and represents the most common etiology of this condition. For a portal-hypertension trait label, assigning LC samples to the control group substantially alters the statistical composition of the dataset. The resulting classification creates a dataset with only 18 cases (INCPH alone) versus 36 controls (HNL and LC combined), whereas the benchmark phenotype definition yields 40 cases (both INCPH and LC) versus 14 controls (HNL only). This reversal in the case-control ratio would substantially change the downstream analysis. Rather than identifying genes associated with portal hypertension as defined in the benchmark, the analysis would instead detect genes that distinguish INCPH from a heterogeneous mixture of healthy and cirrhotic livers, making the result difficult to interpret for the intended task.

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 substantially reduces statistical power. The decision to exclude rather than include LC samples as portal-hypertension cases suggests a failure to align the preprocessing logic with the benchmark phenotype definition.

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 phenotype interpretation errors 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 Summary

AutoGen, a popular multi-agent conversation framework developed by Microsoft (Wu et al., 2023), 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. In our preliminary trials, AutoGen did not produce a complete, valid preprocessing output for the tested GEO cohorts under either the general prompt or the detailed prompt. The failures were consistent with the broader limitation observed across general-purpose agents: the framework could coordinate subtasks, but it did not reliably enforce the domain-specific checks needed for phenotype interpretation, probe-to-gene mapping, and final output validation.

Because the AutoGen runs did not yield concise, auditable execution traces suitable for reproduction in the appendix, we do not include reconstructed code snippets or conversational examples here. We therefore use the AutoGen results only as qualitative evidence that general multi-agent orchestration alone is insufficient for this setting, rather than as a separate quantitative baseline for end-to-end raw-data preprocessing.

A.4 MetaGPT Summary

MetaGPT, a multi-agent framework emphasizing structured role-based collaboration (Hong et al., 2024), 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. In our preliminary trials, MetaGPT did not complete the standardized GEO preprocessing workflow with outputs that satisfied the benchmark requirements. The observed failures reflected the same pattern seen in other general-purpose systems: role-based decomposition and generated documentation did not ensure that all required biological and technical checks were performed.

As with AutoGen, we omit detailed MetaGPT code examples because the available run artifacts are not suitable for concise verbatim presentation in this appendix. We retain the qualitative observation because it motivates GenoMAS’s emphasis on explicit action units, code review, and domain consultation, but we do not treat MetaGPT as a

quantitative baseline for end-to-end raw-data preprocessing.

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 is inconsistent with the benchmark’s binary phenotype definition for portal hypertension:

```
# 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 classification that is misaligned with the benchmark phenotype definition and cannot be used directly to identify genes associated with portal hypertension in this task.

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 grouping LC with INCPH under the benchmark's portal hypertension label. This persistent mismatch shows that strong code generation capabilities alone are not sufficient to align outputs with the task's phenotype definition without explicit procedural guidance.

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, including 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. The reliable phenotype labels are available in the sample-level source name and title fields (for example, IPAH Lung versus Control Lung), while individual IDs alone are not sufficient for case-control assignment.

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 when it is applied only to the individual-ID field, where sample characteristics contain values such as 'individual: AH014' or 'individual: UC003'. These IDs do not reliably encode disease status, so phenotype extraction must instead inspect sample-level source or title fields that explicitly distinguish IPAH from control lung samples.

The human expert implementation uses explicit sample-level labels rather than inferring disease status from individual ID prefixes:

```
def convert_trait(source_name: str,
                  sample_title: str) -> Optional[int]:
    """
    Converts sample metadata to a
    binary trait value (IPAH vs
    Control).
    - 1: IPAH patient (case)
    - 0: Control patient
    The distinction is read from
    explicit source-name/title
    labels.
    """
    text = f"{source_name} {sample_title}".lower()
    if "ipah" in text:
        return 1
    elif "control" in text:
        return 0
    else:
        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 metadata structure used in real genomic datasets. The agent fails to examine all available sample-level fields and therefore misses the explicit IPAH/control labels. This results in complete data loss if no samples can be properly classified from the field selected by the agent.

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 (Shen et al., 2026), 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 ab-

sence 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 = i
            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:

```
# Incomplete clinical characteristic extraction
if 'clinical_group' in data['characteristics']:
```

```
trait = data['characteristics']['clinical_group']
if trait == 'all PAH': #
    Incomplete mapping <-
    trait = 1
elif trait == 'FD':
    trait = 0
# Missing: consistent handling of donor/control samples and PAH subtypes <-
```

The clinical metadata for this dataset includes both `clinical_group` and subtype information, so robust preprocessing must use these fields consistently to distinguish PAH cases from donor/control samples. The trait mapping logic above only handles a narrow subset of labels and does not provide a complete, validated case-control conversion. This results in data loss and potential misclassification of disease or control 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: Misalignment with the Benchmark Phenotype Definition

Under the detailed prompt condition with explicit guidelines, Manus demonstrates an even more concerning failure: a phenotype-labeling error that persists despite detailed instructions. Processing the GSE77627 portal hypertension dataset, Manus

implements a trait classification that is inconsistent with the benchmark definition:

```
# Manus's trait conversion with
detailed instructions
def convert_trait(value):
    value = value.split(":")[-1].
        strip().strip("\n")
    if value == "INCPH":
        return 1
    elif value == "LC" or value == "
HNL": # Benchmark-labeling
        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), which is inconsistent with the portal-hypertension label used in the benchmark. Liver cirrhosis accounts for approximately 90% of portal hypertension cases in Western countries and is the most common cause of this condition. Under the benchmark definition, assigning LC samples to the control group creates a dataset where a substantial fraction of portal-hypertension cases are labeled as controls.

The benchmark phenotype classification, implemented by the human expert, recognizes both disease subtypes:

```
# Human expert: benchmark phenotype
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 label assignment is substantial: Manus's approach yields only 18 cases (INCPH) versus 36 controls (HNL and LC combined), while the benchmark classification produces 40 cases versus 14 controls. This reversal of the case-control ratio 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 phenotype-labeling and bioinformatics preprocessing errors 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 Protocols for the Additional Experiments

This section documents the protocols of the evaluations beyond the main GenoTEX comparison: the benchmark-wide evaluation of a general-purpose coding agent (Section 5.2) and the two external benchmarks (Section 5.4). Two facts hold across all three experiments. First, no agent prompt or predicted output contained gold answers, gold result files, or answer lists. Second, every comparison holds the code-generating model fixed across the methods being compared, so differences reflect agent design rather than the choice of base model.

B.1 General-Purpose Coding Agent on GenoTEX

We drive Claude Code fully end to end: it receives the same GTA task inputs as every method in Table 1 and must proceed from raw data to the final gene list and regression outputs without a task-specific scaffold. The two runs reported in Table 1 use one and the same harness and instruction set; only the backbone differs. The Claude Sonnet 4 run uses extended thinking with the same 8,192-token budget as GenoMAS’s programming agents and the qualitative studies in Appendix A; the Claude Sonnet 5 run uses adaptive thinking at high effort. Metrics and the success-rate definition (code executes without error and writes results to the required path and format) are identical to Table 1.

Because both runs share the same harness, Sonnet 5’s near-perfect execution success (99.42%) indicates that the setup is adequate for producing valid outputs; the low success rate with Sonnet 4 reflects backbone capability on this long, multi-step pipeline rather than a harness artifact. On the auxiliary trait prediction task, Claude Code obtains 9.40% accuracy with Sonnet 4 and 75.32% with Sonnet 5, against GenoMAS’s 86.25%.

B.2 ScienceAgentBench

Benchmark and metrics ScienceAgentBench (SAB) (Chen et al., 2025b) poses real, data-driven scientific tasks: each provides a natural-language goal, a dataset, and an output contract, and the agent must write and run a Python program that produces a specified artifact (a figure, table, trained model, or metric). We evaluate all 27 tasks in the

Table 3: **Success rates on the 27 verified Bioinformatics tasks of ScienceAgentBench**, with 95% Wilson intervals. CBS is SAB’s secondary code-similarity metric. All methods are scored by the same evaluator.

Method	Successes	SR(%), 95% CI	CBS
GenoMAS + repair/retraction	17 / 27	63.0 [44.2, 78.5]	0.870
HAL SAB Self-Debug (public trace)	12 / 27	44.4 [27.6, 62.7]	0.854
Direct (matched coder)	10 / 27	37.0 [21.5, 55.8]	0.816
Self-debug (matched coder)	8 / 27	29.6 [15.9, 48.5]	0.790
HAL Generalist Agent (public trace)	7 / 27	25.9 [13.2, 44.7]	0.815

verified Bioinformatics split. The primary metric is Success Rate (SR), the fraction of tasks whose artifact passes that task’s own evaluator. CodeBERTScore (CBS) is secondary: in SAB’s implementation, a successful program is assigned CBS of 1.0, so aggregate CBS is partly coupled to SR and is not an independent measure of correctness. Every method is scored by the same local SAB evaluator (the standard SAB Docker path); the 16 tasks requiring judgment of a figure use a fixed GPT-4o visual judge applied identically to all methods.

Methods GenoMAS uses GPT-5 for planning and review, Gemini 2.5 Pro as the Domain Expert, and Claude Sonnet 4.5 to write and revise code, retaining its native execution-guided repair and retraction (Section 4). The two locally built controls match the coder: Direct is a single Claude Sonnet 4.5 generation with no scaffold or repair, and Self-debug adds up to two execution-repair iterations without a planner, expert, or reviewer. The two public baselines are recovered from the Holistic Agent Leaderboard (Kapoor et al., 2026), which releases the programs some agents generated on SAB: HAL SAB Self-Debug (SAB’s benchmark-specific self-debug scaffold with Claude Sonnet 4.5) and HAL Generalist Agent (a benchmark-agnostic tool-using agent backed by o4-mini); we re-score both under our common evaluator. During development, only public task statements, output and evaluator contracts, execution failures, and pass/fail outcomes were used.

Statistical notes The 95% intervals in Table 3 are Wilson intervals over the 27 tasks, and the McNemar tests are paired across tasks. Because GenoMAS’s repair loop observed pass/fail outcomes on these tasks, the intervals are descriptive of this task set rather than held-out-test guarantees. The Self-debug control also receives execution feedback, so the contrast isolates GenoMAS’s guided multi-agent repair and retraction from generic self-

Table 4: **HMS on the 11 discovery-requiring DiscoveryBench biology tasks**, averaged over three judge runs. All methods use Claude Sonnet 4.5 as the coding backbone and are scored by the same judge configuration.

Method	HMS
GenoMAS (full)	29.4
w/o guided planning	11.1
w/o code review	30.3
Self-debug	13.6
DiscoveryBench official coder agent	8.1
Direct (no analysis)	1.5

debugging, not repair from no repair.

B.3 DiscoveryBench

Benchmark and metrics DiscoveryBench (Majumder et al., 2025) evaluates data-driven discovery: each task provides a dataset, a metadata file, and a natural-language query, and the agent must analyze the data and state a hypothesis in natural language. The official metric, Hypothesis Matching Score (HMS, 0 to 100), uses an LLM judge to compare the stated hypothesis to the gold hypothesis over three facets (context, variables, and their relationship) and returns a partial-match score; since HMS is noisy, we report the mean of three judge runs per task. Our task set comprises the 20 public biology tasks: 10 from `evolution_freshwater_fish` and 10 from `introduction_pathways_non-native_plants`, the two biology domains in the public split.

Judge configuration DiscoveryBench’s original judge (gpt-4-1106-preview) was unavailable through our API access, so we run the official evaluator, with its scoring logic unchanged, through Claude Sonnet 4.5 for every method. The judge is well calibrated, scoring the gold hypothesis against itself at 85 of 100. Because the judge model differs from the original paper’s, absolute HMS is comparable only within Table 4, not to DiscoveryBench’s published numbers. Since the coder and judge share a model family, an independent-judge cross-check would be desirable; it was not cleanly available because, in our tests, the evaluator’s strict output format was satisfied reliably only by the Claude judge (gpt-4o was badly miscalibrated, and gpt-5 and Gemini 2.5 Pro returned format-parse errors as judges).

The answer-leakage split Many DiscoveryBench queries state the target variables, and sometimes the direction of the relationship, inside the question, so a closed-book guess that echoes the query already earns a high HMS. To separate the tasks where analysis matters, we split the 20 tasks by whether the no-analysis Direct baseline can already solve them: on the 9 answer-embedded tasks Direct scores about 70 HMS by echoing the query, whereas on the 11 discovery-requiring tasks it collapses to about 1.5 HMS. The split is defined by Direct’s failures, not by GenoMAS’s performance, and GenoMAS solves several of the discovery-requiring tasks outright, confirming they are solvable.

Why guided planning drives the margin On the discovery-requiring questions, the difficulty is analytical direction rather than coding: which variables drive an outcome, and how, once confounders are controlled. The baselines pick an analysis without a planning stage, and a debug loop repairs execution rather than reasoning. GenoMAS’s guided planning lays out a strategy that isolates drivers with a controlled multivariable model, and its Domain Expert reinforces the method, so the stated hypothesis names the variables and relationship that the gold’s controlled analysis reflects. The roughly 16-point margin over Self-debug is far larger than HMS’s single-run noise (about 5 points, further reduced by averaging three runs), and the planning ablation in Table 4 attributes it to planning.

B.4 Inference Configuration

Across all experiments, we use the default decoding settings (temperature, maximum tokens, and stopping conditions) of each LLM API. Extended thinking budgets are stated per experiment above; the GenoTEX experiments in Table 1 use an 8,192-token thinking budget for Claude Sonnet 4 (Thinking). Failed API calls are retried with exponential backoff. The proprietary APIs we use do not support random seeds, and each configuration is evaluated in a single run (see Limitations); the exact date-versioned model identifiers and retry settings are documented in our released code.

C 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.

C.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.

C.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.

C.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.

C.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 con-

straints, 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.

C.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.

C.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.

D 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.

D.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
  GEOparse
- 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

```

D.2 GEO Action Unit Examples

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

D.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

```

D.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

```

D.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

```

D.4 TCGA Action Unit Examples

D.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
```

D.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
```

D.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.

E 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.

E.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.

E.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.

E.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.

E.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.

E.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.

F 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.

F.1 Code Generation and Review Process

F.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.

F.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:

ation:

```

### 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."

```

F.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 incor-

rect 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}
```

F.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.

F.3 Context Management and Quality Assurance

F.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.

F.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.

G 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. This is a diagnostic analysis of a single tracked session, intended to illustrate how

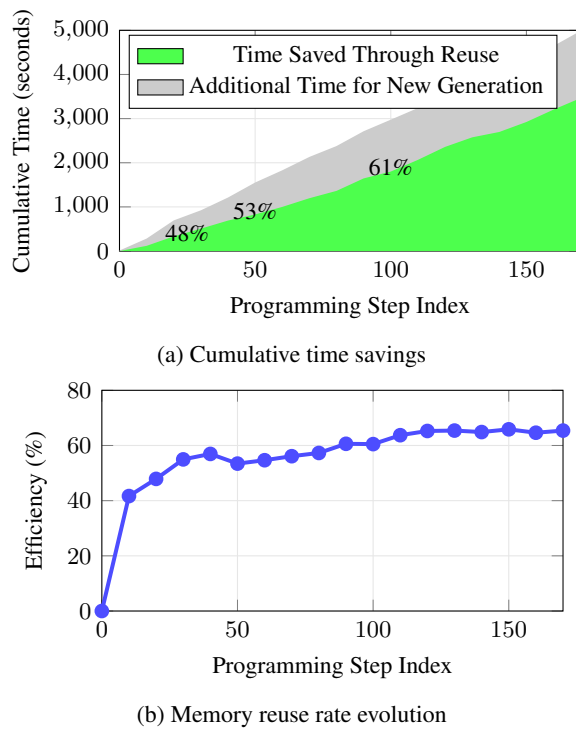


Figure 4: **Memory reuse efficiency in GenoMAS.** (a) Cumulative time savings through memory reuse and (b) memory reuse rate evolution across programming steps. The system rapidly achieves high efficiency, stabilizing around 65% reuse rate after initial learning.

code reuse accumulates, rather than a benchmark-wide efficiency estimate; the benchmark-wide cost and time figures are those in Table 1. 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.

H Exploratory Biological Plausibility: Case Studies

We present exploratory case studies examining genes identified for three complex traits in the unconditional gene-trait association (GTA) problem. For each trait, we assess whether selected GenoMAS genes are biologically plausible in light of established literature. These examples are intended as literature-based plausibility checks rather than independent experimental validation.

H.1 Alzheimer’s Disease

The analysis identified 22 significant genes for Alzheimer’s Disease. Several selected genes have literature support consistent with disease-relevant mechanisms:

GABBR1 (GABA_B Receptor 1) This gene, encoding the GABA_B receptor subunit 1, exhibited the third-highest coefficient (in absolute value). GABA_B receptors play crucial roles in synaptic plasticity and cognitive function. Prior work has reported alterations in GABA_B receptor signaling in Alzheimer’s disease model brains (Salazar et al., 2021). The observed negative association is directionally compatible with this evidence, although the causal interpretation of the coefficient requires further validation.

IGFBP5 (Insulin-like Growth Factor Binding Protein 5) IGFBP5 was selected by GenoMAS and modulates IGF-1 signaling, which is critical for neuronal survival and synaptic maintenance. Experimental studies indicate that IGFBP5 is associated with amyloid pathology and cognitive impairment (Rauskolb et al., 2022), consistent with potential involvement in neurodegenerative processes.

VGF (VGF Nerve Growth Factor Inducible) VGF, a neuropeptide precursor, showed negative association with disease risk. This gene is highly expressed in neurons and plays roles in synaptic plasticity and neurogenesis. Reduced VGF levels have been reported in Alzheimer’s cerebrospinal fluid (Carrette et al., 2003), making it a biologically plausible candidate for follow-up.

PAWR (PRKC Apoptosis WT1 Regulator) PAWR, also known as Par-4, is a pro-apoptotic protein with documented involvement in neurodegenerative diseases. The observed positive association with Alzheimer’s risk is consistent with studies showing elevated PAWR expression in Alzheimer’s brains and its role in mediating neuronal apoptosis

in response to amyloid-beta toxicity (Guo et al., 1998; Xie and Guo, 2005).

These examples span key pathological mechanisms including GABAergic dysfunction, insulin signaling dysregulation, synaptic impairment, and apoptotic processes.

H.2 Lung Cancer

The analysis identified 1,269 significant genes for lung cancer, reflecting the broad transcriptomic changes associated with this malignancy. Several selected genes have clear prior links to lung cancer biology:

TERT (Telomerase Reverse Transcriptase)

TERT is one of the most well-established lung cancer susceptibility genes. TERT encodes the catalytic subunit of telomerase, and polymorphisms in the TERT locus are among the most robust genetic risk factors for lung cancer across multiple populations (McKay et al., 2008; Rafnar et al., 2009). TERT activation is a hallmark of cancer, enabling unlimited replicative potential.

CYP1A1 (Cytochrome P450 Family 1 Subfamily A Member 1) CYP1A1 metabolizes polycyclic aromatic hydrocarbons and other tobacco smoke carcinogens. CYP1A1 polymorphisms have been extensively studied in lung cancer susceptibility, particularly in smoking-related cancers (Shi et al., 2008; Chen et al., 2011). The enzyme's role in bioactivation of procarcinogens makes it a key determinant of individual cancer risk.

RET (Ret Proto-Oncogene) RET alterations, particularly fusions, are established oncogenic drivers in lung adenocarcinoma, occurring in approximately 1-2% of cases (Kohno et al., 2012; Lipson et al., 2012). RET-targeted therapies have shown clinical efficacy, validating its role as a therapeutic target.

These examples span cancer-relevant processes including telomere maintenance, carcinogen metabolism, and oncogenic signaling.

H.3 Hypertension

The analysis identified 108 significant genes for hypertension. Several selected genes are connected to vascular biology and blood-pressure regulation:

TGFB2 (Transforming Growth Factor Beta 2)

TGFB2 exhibited negative association with hypertension risk. TGF- β signaling plays complex roles

in vascular remodeling and blood pressure regulation. Large-scale genome-wide association analyses have linked TGFB2 or nearby regulatory loci to blood pressure traits (Evangelou et al., 2018), and the protein influences vascular smooth muscle cell differentiation and arterial stiffness.

RHOA (Ras Homolog Family Member A)

RHOA, encoding a small GTPase, showed negative association with hypertension. RhoA/Rho kinase signaling is a well-established regulator of vascular tone and contractility. Enhanced RhoA activity increases vascular smooth muscle contraction and contributes to hypertension pathogenesis (Loirand et al., 2006; Loirand and Pacaud, 2010). The coefficient direction should be interpreted cautiously because expression-level associations do not directly measure pathway activity.

CYP1B1 (Cytochrome P450 Family 1 Subfamily B Member 1)

CYP1B1 metabolizes endogenous compounds including estrogens and arachidonic acid, generating vasoactive metabolites. Genetic and experimental evidence implicates CYP1B1 in blood-pressure regulation and angiotensin II-driven hypertension (Jennings et al., 2010; Malik et al., 2012).

EGR1 (Early Growth Response 1) EGR1, a transcription factor, showed negative association in our analysis. EGR1 is rapidly induced by various stimuli and regulates genes involved in vascular inflammation, smooth muscle proliferation, and fibrosis (Khachigian et al., 1996; Khachigian, 2006). Its dysregulation contributes to vascular remodeling and end-organ damage.

ANGPT2 (Angiopoietin 2) ANGPT2 regulates angiogenesis and vascular stability. Elevated circulating ANGPT2 has been reported as a marker of endothelial dysfunction and adverse cardiovascular outcomes in patient populations with chronic kidney disease or on dialysis (David et al., 2009; Shroff et al., 2013). While these reports are not hypertension-specific, they place ANGPT2 within the vascular-pathology axis relevant to blood pressure regulation, and the observed positive association warrants further investigation.

These examples are involved in multiple hypertension-relevant pathways: vascular tone regulation, smooth muscle contractility, endothelial function, metabolic control, and inflammatory responses.

H.4 Summary

Across all three traits, selected genes show literature-consistent biological plausibility and span pathophysiological mechanisms relevant to each disease. These include neurotransmitter signaling and neurodegeneration for Alzheimer's Disease, carcinogenesis and oncogenic signaling for lung cancer, and vascular biology and blood pressure regulation for hypertension. The results encompass both well-established disease genes (e.g., TERT for lung cancer) and biologically plausible candidates that merit further investigation, supporting the use of GenoMAS outputs as hypotheses for follow-up genomic analysis.

I 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.

I.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.

I.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.

I.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.

I.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.

I.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.")
```

I.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.

I.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.

I.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
```

I.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.

I.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.

J Qualitative Analysis of Clinical Feature Extraction Failures

Section 5.3 shows that clinical feature extraction is the hardest stage of the pipeline. GenoMAS reaches a composite similarity correlation (CSC) of 32.61% on this sub-task, far below the 91.15% it attains on gene expression data. That number is the highest among all systems we evaluated (Figure 3(b)), yet it also makes clear that the task is far from solved. To understand the remaining gap, we read through GenoMAS trajectories and inspected the cohorts where the extracted trait diverged from a defensible phenotype definition. The errors are rarely arbitrary. They cluster into a small set of recurring patterns, each rooted in a specific way

that free-text GEO metadata can be misread when it is mapped onto an exact case-control label.

One observation frames the rest of this section. The mechanical steps of preprocessing, such as parsing characteristic rows and binarizing values, almost always run to completion, consistent with our overall execution success rate of 98.78% in Table 1. What remains difficult is the judgment layered on top of those steps: whether a field encodes the target phenotype at all, whether a measured value reflects disease status or merely treatment response, and whether the cohort even studies the trait in question. These judgments are where domain knowledge matters most, and they are where the residual errors concentrate. We describe four patterns below, using excerpts from audited notebooks rather than full execution logs. For each case we state what made the cohort difficult, the decision the agent made, why it is problematic, and the behavior we would consider correct.

J.1 Treating a within-patient contrast as case versus control (GSE71994, Hypertension)

This cohort profiles peripheral blood mononuclear cells from twenty hypertensive patients, all receiving standardized antihypertensive medication, with each patient sampled twice. The study describes its goal as separating controlled from uncontrolled hypertensives, and it records each sample's systolic blood pressure. The cohort therefore contains no normotensive controls: every subject has hypertension.

The agent selected the blood pressure field as the trait and binarized it at 140 mmHg.

```
trait_row = 6 # "systolic blood
               pressure in a sample"
def convert_trait(value):
    """systolic BP >= 140 mmHg
       indicates hypertension."""
    bp = int(value.split(":", 1)[1].
              strip())
    return 1 if bp >= 140 else 0 #
                                below 140 is labeled 0, "
                                normal" <-
```

The threshold is reasonable for screening untreated individuals. The problem is the population it is applied to. In a cohort of treated hypertensives, a reading below 140 mmHg means the hypertension is controlled by medication, not that the patient is healthy. Labeling these samples as 0 quietly redefines the control group as well-managed cases. The agent's own note records the cohort as patients "classified as controlled (<140 mmHg) or uncontrolled ($\geq$ 140 mmHg)," which shows that it read

the design correctly. The conversion nonetheless mapped controlled patients onto the healthy-control label. The result, sixteen positives and twenty-four negatives, passed the distribution check and was saved as usable, so the mislabeling propagated to the association step. A model trained on this table contrasts high against controlled blood pressure within treated patients, not hypertension against health. A faithful handling would recognize that the cohort lacks normotensive controls, and would either mark the trait as unsuitable for a case-control contrast or frame the contrast explicitly as treatment response.

J.2 Forcing a trait from a cohort that does not study the target condition (GSE87629, Creutzfeldt-Jakob Disease)

This cohort was retrieved as a candidate for Creutzfeldt-Jakob disease, a prion neurodegenerative disorder. Its metadata, however, describes a celiac disease study. Every subject is labeled as a celiac patient, the disease state is “biopsy confirmed celiac disease,” and the design compares a gluten challenge against a control diet. The only quantitative clinical field is the villus height to crypt depth ratio, a histological measure of intestinal mucosal damage used in enteropathy.

Asked to extract a Creutzfeldt-Jakob disease trait, the agent selected this intestinal ratio and passed it through as the label.

```
trait_row = 5 # "biopsy data, villus
height to crypt depth" <-
def convert_trait(value):
    """Higher values indicate
    healthier intestinal tissue.
    Lower values indicate more severe
    celiac disease activity."""
    return float(value.split(':')[1].
strip()) # used as the CJD
label
```

The villus to crypt ratio has no established role as a phenotype for Creutzfeldt-Jakob disease, and the docstring the agent wrote even names celiac disease activity while the label is assigned to the neurological trait. The deeper issue is not the conversion but the missing relevance check. The agent treated the assigned trait name as a given and looked for any field it could bend into a label, rather than first asking whether the cohort studies the target disease at all. Setting `is_trait_available` to true here encodes a phenotype the dataset does not contain. The correct behavior is to recognize that a celiac disease cohort carries no Creutzfeldt-Jakob disease phenotype, and to mark the trait as unavailable so

that an irrelevant cohort is kept out of the analysis.

J.3 Entangling the trait with a covariate and overlooking species (GSE44147, Longevity)

This cohort is a developmental time series of mouse prefrontal cortex. The strain is recorded as C57BL/6 and the ages span 2 to 904 days. GenomEX targets human transcriptomics, so a mouse cohort is already outside the intended scope. The trait choice compounds the issue. The agent set both the trait and the age covariate to the same characteristic row, and defined longevity as a threshold on age.

```
trait_row = 2 # age in days
age_row = 2 # the same field, also
used as the age covariate <-
def convert_trait(value):
    """C57BL/6 mice typically live
    2-3 years."""
    days = int(value.split(':')[1].
replace('days', '').strip())
    return 1 if days > 365 else 0
def convert_age(value):
    return int(value.split(':')[1].
replace('days', '').strip())
```

Because the trait is a deterministic function of age, the label and the age covariate carry the same information, and any model that adjusts for age while predicting this trait is degenerate. Setting aside the statistics, age at sampling is not longevity. The cohort records when tissue was collected across development, not how long each animal lived. The agent’s comment that the threshold is appropriate for mice shows that it noticed the species, but it proceeded regardless. The trajectory that followed never settled on a coherent trait and repeated the clinical analysis step without converging. The appropriate response is to treat this cohort as out of scope for a human longevity study, rather than to manufacture a binary label from the age axis.

J.4 Synthesizing labels when extraction fails (GSE90074 and GSE87005, Atherosclerosis)

The most consequential failures occur when the agent cannot read the clinical fields at all, and then fills the gap instead of stopping. In GSE90074, the parsed sample characteristics collapsed to a single technical annotation, “pool of human cell line RNA,” so the real per-sample phenotypes were not reachable through the standard characteristic rows. Rather than report the trait as unavailable, the agent generated placeholder labels at random.

```
# Since we don't have valid clinical
data, create a synthetic trait
column
random.seed(123)
case_samples = random.sample(
    sample_ids, int(0.4*len(
        sample_ids))) # random <-
synthetic_clinical_df.loc[trait,
    case_samples] = 1
synthetic_clinical_df.loc['Age'] =
    [random.randint(40, 80) for _ in
        sample_ids]
synthetic_clinical_df.loc['Gender'] =
    [random.randint(0, 1) for _ in
        sample_ids]
```

These labels bear no relation to any patient. Linking them to the real expression matrix yields a table whose associations are noise by construction, and because the random split is not degenerate it passed the distribution check and was saved as usable. We observed this fallback in only two cohorts, GSE90074 and GSE87005, so it is rare. In both, the agent recorded the substitution in its note, stating that “synthetic clinical data was generated for testing purposes only,” which means the system’s own logging surfaces these cases for human audit. The behavior is nonetheless unsafe. A failed extraction should fail closed, marking the trait as unavailable, and should never fill it with invented values. This points to a simple and high-value safeguard: an explicit rule that forbids synthetic labels and routes any unreadable clinical structure to the not-available path.

J.5 Implications for future work

Taken together, these cases show that the clinical extraction gap is less about code and more about clinical judgment under ambiguous metadata. Three lessons stand out. First, a measured value is not a diagnosis. Blood pressure, symptom scores, and histology readings describe state or severity, and mapping them onto a case-control label requires knowing the population in which they were measured. Second, the trait name cannot be taken for granted. Before extracting a label, the system should confirm that the cohort actually studies the target condition and is of the right species. Third, when the clinical signal is genuinely absent or unreadable, the safe action is to abstain rather than to approximate.

Each lesson maps onto a concrete extension of GenoMAS rather than a redesign. The planning framework already represents trait-availability decisions and supports retraction, so these checks fit naturally as added guidelines and review cri-

teria: a case-control validity check that verifies controls exist, a relevance check that compares the cohort’s described condition against the target trait, and a hard guard against synthetic labels. We view these as the most promising near-term directions for raising clinical extraction quality. Even without them, GenoMAS already attains the strongest performance on this sub-task among all evaluated systems, and the remaining gap reflects what we consider one of the central open problems in automating transcriptomic analysis, namely the reconciliation of unstructured clinical text with exact biological definitions.

K 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.

K.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.

K.2 INFO-Level Notes: Documenting Standard Processing

INFO notes constitute the majority of agent observations, documenting successful preprocessing steps and dataset characteristics. These notes pro-

vide audit trails for quality assurance and enable researchers to understand processing decisions.

K.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.

K.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 (Pulmonary arterial
hypertension):
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.

K.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.

K.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.

K.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
      .
```

K.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.
```

K.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.

K.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.

K.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).
```

K.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.
```

K.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.
```

K.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.

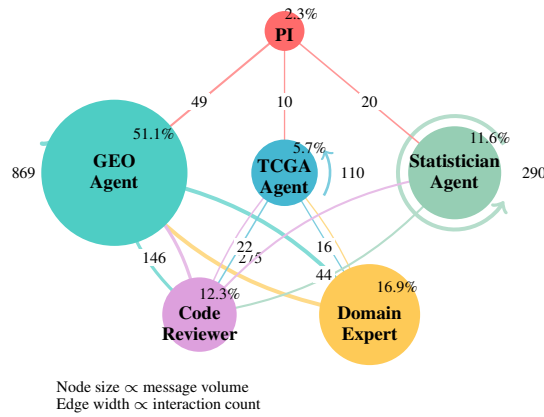
System Enhancement Opportunities Recurring error patterns inform development priorities. For example, the prevalence of probe-to-gene mapping failures led to enhanced synonym dictionaries and platform-specific mapping strategies in subsequent GenoMAS versions.

K.6 Summary

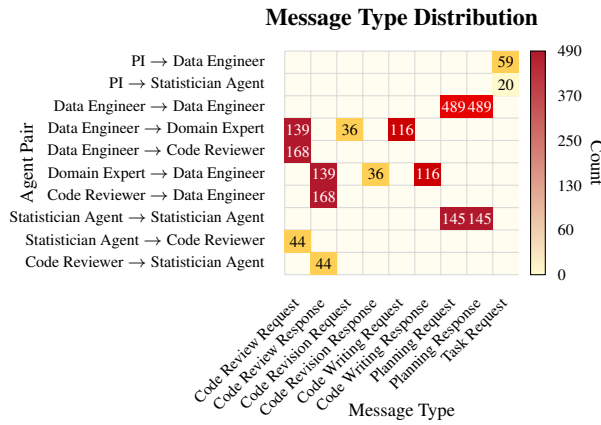
The agent-generated notes demonstrate GenoMAS's ability to provide transparent, interpretable preprocessing workflows. By systematically documenting observations across severity levels, the system enables efficient human oversight while

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.

L Multi-agent Collaboration Patterns



(a) Agent collaboration network



(b) Message type distribution heatmap

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

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 (Hong and Page, 2004) as manifested in our heterogeneous LLM architecture. The PI’s low involvement demonstrates the system’s high degree of autonomy (97.7% self-coordinated interactions), which contributes to a 44.7% API cost reduction over prior methods.

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

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