

MARWA: MULTI-AGENT RETRIEVAL-AUGMENTED FRAMEWORK FOR RELIABLE BIOINFORMATICS WORKFLOW AUTOMATION

Anonymous authors

Paper under double-blind review

ABSTRACT

The rapid growth of multi-omics data has driven the expansion of bioinformatics analysis tools. Common bioinformatics tasks often rely on workflows, which link multiple tools into structured pipelines for reproducibility and scalability. Yet, building workflows manually is slow and error-prone, motivating efforts toward automation. However, bioinformatics workflow automation remains difficult due to the need to clarify vague analytical objectives, coordinate heterogeneous tools, and generate intricate tool commands. Despite the potential of large language models (LLMs) to aid bioinformatics workflow recommendation through advanced semantic understanding and logical reasoning, current agent frameworks often rely on one-shot generation, weak tool retrieval solution, and limited evaluation scheme, resulting in fragile workflow automation. We propose **MARWA**, a Multi-Agent Retrieval-augmented framework for reliable bioinformatics Workflow Automation. The framework emphasizes a step-by-step generation process with error handling at each stage to ensure robustness. We introduce a retrieval-augmented framework to strengthen tool command accuracy, which incorporates multi-perspective LLM-augmented descriptions and employs contrastive learning. We further design a two-stage evaluation framework, combining expert-verified execution on 40 curated tasks with large-scale benchmarking on 2,270 tasks using LLM-based evaluation. Our experiments demonstrate that MARWA consistently outperforms baselines in pass rate, workflow quality and scalability. Our work provides a foundation for trustworthy bioinformatics workflow automation.

1 INTRODUCTION

Bioinformatics is an interdisciplinary field that combines computational science, statistics, and biology to analyze large and complex biological datasets through computational and statistical methods (Luscombe et al., 2001; Gauthier et al., 2019; Baxevanis et al., 2020). With the advances in high-throughput biological technologies (Rhoads & Au, 2015), the field is now confronted with a rapid expansion of biological data. This explosive growth has spurred the development of numerous bioinformatics tools, covering diverse fields such as genomics (Lesk, 2017; Bustamante et al., 2011; Lips et al., 2022), structural biology (Orlando et al., 2022; Jones & Thornton, 2022) and evolutionary biology (Sober, 1994; Losos et al., 2013). These tools have further enabled significant advances in personalized medicine (Heinken et al., 2023) and drug discovery (Hemmerling & Piel, 2022).

Due to the diverse requirements for analyzing biological data, such as genome assembly (Sohn & Nam, 2018) and differential expression analysis (Anders & Huber, 2010), these tasks cannot be accomplished using a single bioinformatics tool alone. Instead, they depend on multi-step workflows that organize bioinformatics tools in a sequential, flow-based manner (Fig 1).

Traditional workflow construction often relies heavily on manual scripting and command-line operations. With the emergence of new technologies and algorithms, the workflows are getting increasingly complicated (Subramanian et al., 2020; Schlotter et al., 2018). This approach is not only time-consuming and prone to errors but also hinders repeatability. These issues highlight the need for more automated, intelligent, and trustworthy methods to create bioinformatics workflows.

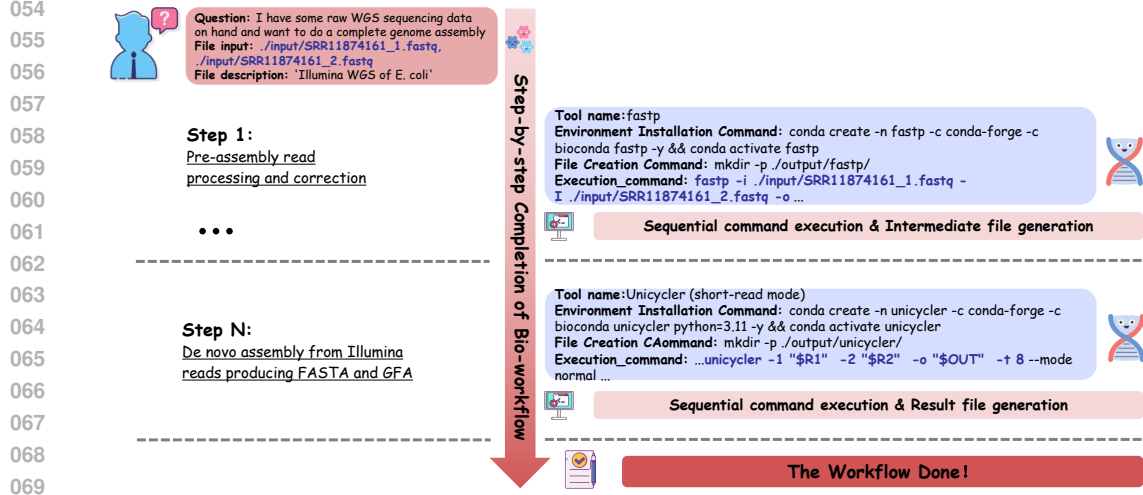


Figure 1: A bioinformatics workflow example for de novo genome assembly. Raw Illumina paired-end reads are processed and quality-controlled (e.g., with fastp) before being assembled into contigs (e.g., with Unicycler), producing final outputs such as FASTA and GFA files. This illustrates the pipeline nature of bioanalysis, where specialized tools are chained together, with the output of one step forming the input to the next.

Recently, large language models (LLMs) (Zhao et al., 2023; Park et al., 2023; Xi et al., 2025), with their advanced semantic understanding and logical reasoning capabilities, are opening new possibilities for automating bioinformatics workflows. Systems such as AutoBA (Zhou et al., 2023) and BioMaster (Su et al., 2025) show the potential of AI-driven agents, demonstrating their capabilities in automating bioinformatics workflows. However, these approaches remain constrained in three key aspects:

- Adopting one-shot generation strategies makes it struggle to handle vague analytical objectives, coordinate heterogeneous tools, and generate intricate command-line specifications.
- The lack of robust semantic representations for bioinformatics tools makes it difficult to retrieve relevant tools during the retrieval-augmented generation (RAG) (Lewis et al., 2020) process.
- The absence of rigorous evaluation framework results in insufficient validation of the generated workflows’ reliability and reproducibility.

To address these challenges, we propose a **Multi-Agent Retrieval-augmented framework** for reliable bioinformatics **Workflows Automation (MARWA)**. Our work makes three key contributions:

- We propose MARWA, a step-by-step multi-agent framework that leverages historical context at each stage of workflow construction, thereby enhancing the flexibility and robustness of bioinformatics workflow automation.
- We design a RAG framework that integrates multi-perspective LLM-enhanced tool descriptions with contrastive representation learning, producing discriminative embeddings that significantly improve tool retrieval accuracy and command generation reliability.
- We construct two representative datasets and evaluation standard for bioinformatics workflow automation, comprising a few-shot executable dataset, a large-scale dataset with 2,270 high-quality workflow queries and establish a two-stage evaluation scheme that combines human execution with LLM proxy evaluation to ensure rigorous and reproducible benchmarking.

2 RELATED WORK

The automation of bioinformatics workflows has undergone a steady evolution, moving from manual construction to intelligent recommendation and, more recently, to LLM-driven automation.

Workflow Management Systems Early advances were supported by workflow management platforms such as Galaxy (Jalili et al., 2020)¹, Snakemake² and Nextflow (Langer et al., 2025)³, which provide standardized execution environments and improve reproducibility. Despite these contributions, workflow design still mainly requires manual tool selection and scripting, which renders the process inefficient and prone to mistakes.

Tool Recommendation To reduce the burden of tool selection, recommender systems were proposed. For instance, Kumar et al. (2021) employed gated recurrent units (GRU) (Dey & Salem, 2017) neural network in Galaxy to capture higher-order dependencies among tools. Green et al. (2024) further extended this idea by representing workflows as graphs and applying graph neural networks (Wu et al., 2020) with semantic embeddings of tool descriptions. These approaches improve context-aware tool discovery but remain limited to tool-level assistance rather than full workflow automation.

LLM-Based Workflow Automation The recent emergence of LLMs has enabled more comprehensive automation (Xi et al., 2025; Zhang et al., 2024; Xiao et al., 2024). AutoBA demonstrated an LLM-based agent can design, implement and execute workflows for diverse omics analyses (Zhou et al., 2023). However, its single-agent design often led to error accumulation in long and complex pipelines. To address this, BioMaster introduced a multi-agent framework with specialized agents for planning, execution, and debugging, combined with RAG of tool knowledge (Su et al., 2025). This multi-agent design improved adaptability and robustness, yet it still faced limitations in overall accuracy and reliability.

3 METHODOLOGY

3.1 OVERALL ARCHITECTURE

The overall framework of MARWA is illustrated in Fig 2. It is composed of six cooperative LLM-based expert agents—Analyzing, Planning, Selecting, Generating & Executing, Debugging, and Judging—organized into a closed-loop pipeline. The system is further supported by two auxiliary components: (1) a retrieval module that provides reference information about bioinformatics tools, and (2) file system interface access that grounds decisions in the actual execution environment.

The user’s input is defined as (1) a list of input files (including their file name and file path), (2) file descriptions (such as format or sequencing type), and (3) the analytical goal (for example, differential expression analysis).

Each agent processes the input and converts it into a structured output, facilitating subsequent parsing. The general agent workflow, as shown in the Fig 3, is outlined below along with a summary of their roles.

Analyzing The Analyzing agent refines the user input into a structured task specification. It produces descriptions of the input and output files (including their formats and data types), along with a clarified analytical objective. Appendix A.1 for the prompt of the agent.

Planning The Planning agent predicts the next tool to be used in the workflow (see Appendix A.2) based on the refined task and the tools already applied. It provides the tool name, a brief description, and its intended function. It also queries the retrieval module to obtain a set of candidate tools with corresponding descriptions and example commands.

¹<https://usegalaxy.org/>

²<https://snakemake.github.io/>

³<https://www.nextflow.io/>

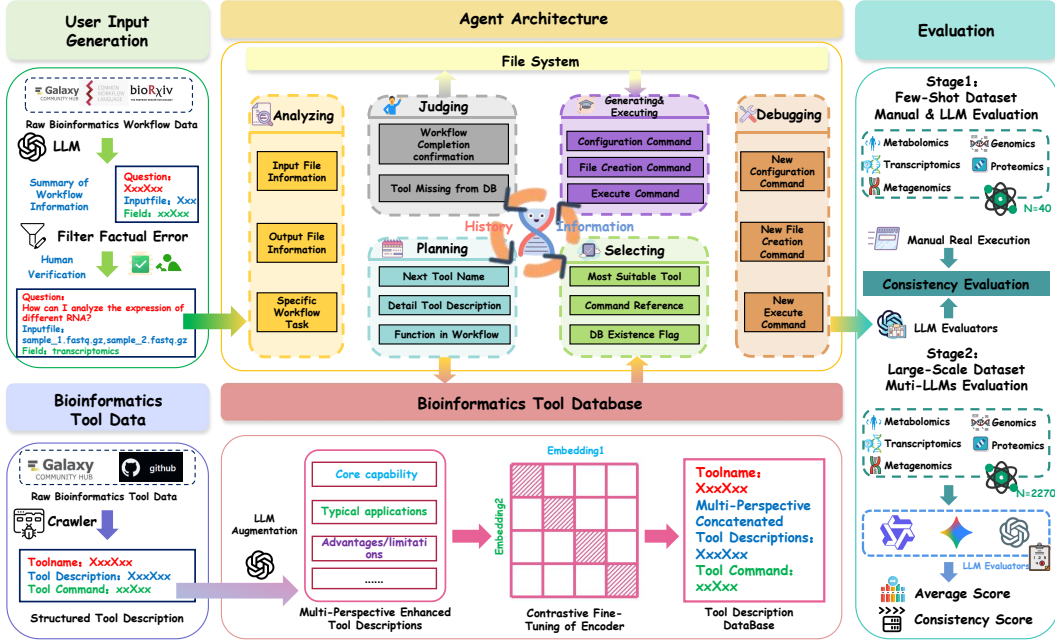


Figure 2: The overall framework of MARWA. The left part shows the generation methods of the data. The middle part illustrates tool retrieval and agent architecture for workflow automation construction. The right part presents workflow evaluation framework.

Selecting The Selecting agent decides whether to adopt one of the retrieved candidate tools or retain the one proposed by Planning. If a retrieved tool is chosen, its description and command template are adopted; otherwise, the Planning output is used. Appendix A.3 for the prompt of the agent.

Generating & Executing The Generating & Executing agent constructs executable commands based on the chosen tool and the information available in the system. These commands include environment setup, directory creation, and the actual execution (detail in Appendix A.4). The commands are then executed, and the success or failure of the execution determines the next step.

Debugging If execution fails, the Debugging agent uses the error message to iteratively refine the command set. This process is repeated until the command succeeds or until five attempts have been made. Appendix A.5 for the prompt of the agent.

Judging The Judging agent evaluates whether the overall analytical task has been completed (detail in Appendix A.6). If the task is incomplete, the system loops back to Planning to select the next tool; if complete, the execution terminates. Tools not covered by the retrieval module but successfully executed are recorded, along with their verified commands, to expand the system’s tool database.

3.2 AUXILIARY COMPONENTS

3.2.1 EMBEDDING

Since bioinformatics tool descriptions often come from heterogeneous platforms and vary widely in length, perspective, and level of detail (Ison et al., 2021), conventional embedding methods struggle to achieve precise semantic alignment. To improve the accuracy and reliability of MARWA in the tool retrieval phase, we design a multi-perspective LLM-augmented strategy and refined through contrastive learning fine-tuning.

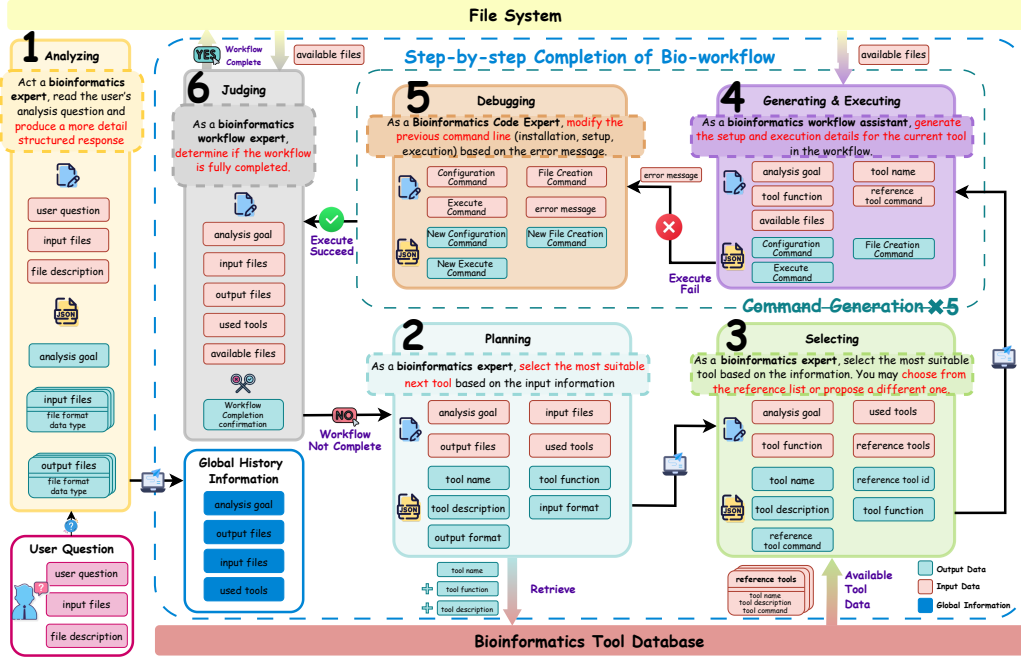


Figure 3: Workflow of MARWA’s specialized agents, illustrating the input data, intermediate processing steps, and final output generation.

Data Sources We collected raw tool data from open repositories such as Galaxy and GitHub⁴. These unformatted text data are parsed into structured fields including tool name, description, command and parameters. By removing duplicate entries based on names and command hashes, followed by manual verification we have built a curated database containing 3,148 unique tools. This provides a solid foundation for the subsequent stages.

LLM-based Multi-perspective Enhancement To enrich the semantic information of the bioinformatics tools, we employ LLM to generate a diverse range of descriptions for each tool, covering perspectives such as its core capabilities and typical applications (Prompt in Appendix A.7). The resulting augmented descriptions increase semantic diversity and provide complementary views of the same tool, which are stored in structured form for downstream training.

Contrastive Learning We adopt BERT (Devlin et al., 2019; Liu et al., 2019) as the encoder, representing each tool description with the [CLS] embedding. We select BERT not only because it remains a widely recognized and reproducible baseline, but also because it provides stable and efficient fine-tuning. This allows us to highlight that the improvements mainly come from our augmentation and contrastive framework rather than from a stronger backbone. Training uses a multi-positive contrastive loss, where augmented descriptions of the same tool serve as positive pairs.

Evaluation The dataset is derived from the enhanced results based on LLM. The experimental results are shown in Table 1. Our embedding method (LAFT) achieves consistent improvements over both general-purpose and domain-specific baselines. These results highlight the effectiveness of combining LLM-based augmentation with contrastive learning in capturing the functional semantics of bioinformatics tools. The contribution of this module will be further validated in subsequent ablation studies.

⁴<https://github.com/>

Table 1: Retrieval Performance of baseline embeddings and LLM-augmented fine-tuned model.

Model	MRR	Hit@1	Hit@3	Hit@10
BERT (Devlin et al., 2019)	0.1988	0.1307	0.2010	0.2261
all_MiniLM_L6_v2 (Wang et al., 2020)	0.4842	0.3920	0.5477	0.5879
Bge m3 (Chen et al., 2024)	0.5466	0.4623	0.5729	0.6432
PubMedBERT (Gu et al., 2021)	0.5798	0.5025	0.6181	0.6533
LAFT	0.6686	0.5779	0.6985	0.7638

3.2.2 FILE SYSTEM INTERFACE

A key challenge in automated workflow generation is bridging the gap between abstract plans generated by LLMs and the real execution environment. To address this, MARWA integrates a file system interface that enables agents to directly query and interact with the underlying directory structure. Specifically, the module provides access to the names and formats of available files, which are then used to guide the construction of subsequent commands.

This interaction yields two main benefits. First, by grounding command generation in the actual file system, the framework reduces errors caused by incorrect file references or incompatible input-output specifications. Second, it ensures that intermediate results are consistently tracked and made available for downstream tools, thereby improving the continuity and robustness of multi-step workflows.

Overall, the file system interface transforms MARWA from a purely symbolic generator into a system that is context-aware and execution-oriented, enhancing both the accuracy of command construction and the reliability of workflow completion.

4 EXPERIMENTS

4.1 DATASETS

To evaluate the effectiveness of our framework, we constructed datasets from multiple real-world bioinformatics workflow repositories, including the Galaxy platform, the Common Workflow Language (CWL)⁵ collection, and preprint articles from BioRxiv⁶. These sources were selected because they represent diverse workflow practices and contain detailed applications, covering a wide range of research domains such as metabolomics (Liu & Locasale, 2017), transcriptomics (Lowe et al., 2017), metagenomics (Wooley et al., 2010), genomics (Lesk, 2017) and proteomics (Aslam et al., 2016), thereby ensuring diversity and representativeness.

For further evaluation, we create two datasets of different scales. Table 2 illustrate the statistics of them. The first is a few-shot dataset containing 40 tasks carefully selected by bioinformatics experts (detail in Appendix B.1). Each task is designed for real execution in practical workflows, allowing manual verification and in-depth inspection of model performance. Despite its small size, this dataset reflects real-world research demands across major bioinformatics domains, and the task distribution aligned to current practices (Mitchener et al., 2025). To generate the second dataset, we leveraged diverse real-world bioinformatics workflow sources described above and prompted a LLM from three distinct roles—lab researcher, clinician, and data engineer. This process produced a large-scale dataset of 2,270 tasks, which has been validated for quality by bioinformatics experts (Appendix A.8 shows the prompt; dataset examples in Appendix B.2). The large-scale dataset maintains a similar domain distribution, further ensuring the representativeness of the small set.

Table 2: Statistics for the datasets.

	Small	Large
metabolomics	4	80
transcriptomics	10	600
metagenomics	8	390
genomics	10	900
proteomics	8	300
total	40	2270

⁵<https://view.commonwl.org/workflows/>

⁶<https://www.biorxiv.org/>

4.2 EVALUATION FRAMEWORK

To objectively assess the capabilities of different models, we adopted a two-stage evaluation framework. In the first stage, we conducted experiments on the small dataset. Each task was executed manually by domain experts and also simulated by LLMs. We then calculated consistency scores between human execution and model outputs, demonstrating that LLMs can provide reliable evaluations of bioinformatics workflows. The expert-executed results on the small dataset serve as the ground truth, primarily establishing the feasibility of the approach. In the second stage, we scaled up to the large dataset and employed multiple LLMs as evaluators. The large-scale evaluation aims at trend validation and demonstrates the scalability of the method. These models were used to assign scores to different workflows, and the final results were reported in terms of both average scores and cross-model consistency. The adoption of LLM-based proxy evaluation is a reasonable yet approximate strategy suitable for large-scale benchmarking. While it confirms the robustness of the method, it does not fully equate to real execution outcomes.

For a comprehensive evaluation, we included both proprietary and open-source models, specifically selecting GPT-4o (GPT-4o) and Gemini-2.5-pro-exp (Gemini2.5) as leading closed-source models, alongside Qwen 2.5 72B-Instruct (Qwen2.5-72B) as a representative open-source alternative. These models were chosen based on prior studies which indicate their strong performance in relevant evaluation tasks (Gu et al., 2024; Liu et al., 2025).

4.3 EVALUATION METRICS

We adopted a combination of human-centered and model-based metrics. For manual execution, we used $h_Pass@n$, which measures the success rate of completing a task within n manual execution attempts.

For LLM proxy evaluation, we developed a structured scoring template that includes six metrics: (1) **Workflow Completion (Comp)**: Measures whether the workflow achieves the analysis goal (0–3; higher is better). (2) **Workflow Redundancy (Redun)**: Measures whether unnecessary or redundant tools are included (0–3; lower is better). (3) **Installation Accuracy (Inst)**: Correctness of tool installation commands (0–2; higher is better). (4) **Path Accuracy (Path)**: Correctness of file paths used in tool commands (0–2; higher is better). (5) **Parameters Accuracy (Param)**: Correctness of command-line parameters (0–2; higher is better). (6) **Executable Flag**: whether this command be executed successfully (True or False). Score criteria can be found in the Appendix Table A.2.

The first two metrics operate at the workflow level (prompt in Appendix A.9), while the last four focus on individual tools (prompt in Appendix A.9). These metrics align with common issues in computational method evaluation, making the overall assessment both rigorous and transparent. If a step fails, the system may invoke Debugging to adjust commands and re-evaluate. We define $m_Pass@n$ as the probability of task success within n LLM-based execution attempts.

To quantify agreement between human and LLM evaluations, we used two measures: (1) **Pass/Fail Agreement Rate (PFAR)**: The proportion of steps where real human execution and LLMs agree on pass/fail outcomes. (2) **Score Agreement Rate (SAR)**: The proportion of instances where the human and LLM scores match exactly for each metric. Detail Formula in Appendix C.2.

We also computed Krippendorff’s alpha (k) (Krippendorff, 2018; 1970) to assess inter-model agreement among LLM evaluators across all five metrics, providing a measure of consistency at both workflow and tool levels. In line with established conventions, values above 0.80 indicate reliable agreement, values between 0.67 and 0.80 are considered tentatively acceptable, and values below 0.67 reflect insufficient consistency.

4.4 FEW-SHOT DATASET VALIDATION

We compared MARWA against three baseline methods: LLM-only, Autoba and Biomaster. All models utilized GPT-4 turbo as the underlying agent to ensure a fair and consistent evaluation. Experimental details can be found in the Appendix C.3.

On the small dataset, results are summarized in Table 3. MARWA surpassed all baseline methods across every evaluation metric, achieving superior performance in both real human execution and

Table 3: The main results of MARWA and different kinds of baselines on the small dataset.

Method	h_Pass@1	h_Pass@2	m_Pass@1	m_Pass@2	PFAR	SAR
LLM-only	0.100	0.100	0.100	0.150	0.900	0.857
Autoba	0.250	0.250	0.350	0.350	0.838	0.828
Biomaster	0.300	0.350	0.375	0.450	0.813	0.819
MARWA	0.375	0.450	0.425	0.475	0.813	0.824

LLM-based simulation. MARWA’s performance advantage can be attributed to its improved capability in selecting appropriate tools, generating more accurate file paths and specifying precise command-line parameters, as clearly demonstrated in the Figure 4). A specific running instance of MARWA is provided in the Appendix D. The moderate performance observed across all methods is primarily due to the inherent complexity of real-world bioinformatics workflow automation, which involve multi-step analytical processes, domain-specific tool integration and stringent parameter tuning requirements.

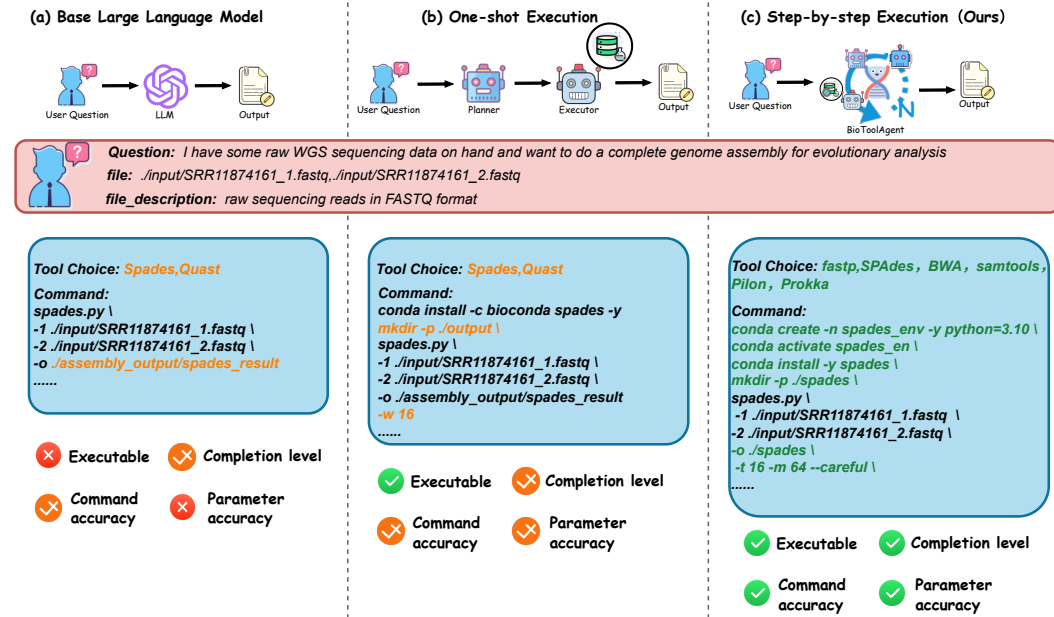


Figure 4: Comparison of bioinformatics workflow automation methods.

4.5 LARGE-SCALE DATASET VALIDATION

On the large dataset, MARWA demonstrates consistent superiority across nearly all evaluation metrics. Result in Table 4. We also have the following findings. (1) **Workflow Completion and Redundancy:** While MARWA achieves strong workflow completion (Comp:2.72), the LLM-only approach attains a higher score (2.76) but with more redundancy (Redun:0.31vs0.15). The LLM-only method relies on redundant tools to superficially improve coverage, whereas MARWA emphasizes precision and efficiency through iterative self-correction. Other baselines perform worse in both metrics due to their inability to revise errors in a single pass, leading to accumulated inaccuracies. (2) **Tool Command-Level Reliability:** MARWA achieves the highest path accuracy (Path:1.78), parameter accuracy (Param:1.27) and installation correctness (Inst:1.75). These metrics reflect MARWA’s ability to generate reliable tool commands, which is critical for real-world execution. By comparison, all the baselines perform poor on the path accuracy due to the absence of real file system interaction. Although Biomaster incorporates RAG, its simplistic embedding mechanism often fails to retrieve relevant and accurate information, resulting in incorrect parameter usage.

Table 4: The main results of MARWA and different kinds of baselines on the large dataset.

Method	Models	Comp	Redun	Inst	Path	Param	m_Pass@1
LLM-only	GPT-4o	2.83	0.29	0.61	0.19	0.76	0.11
	Gemini2.5	2.75	0.31	0.44	0.12	0.72	0.11
	Qwen2.5-72B	2.69	0.32	0.33	0.12	0.68	0.09
	mean/ <i>k</i>	2.76/0.77	0.31/0.77	0.46/0.66	0.14/0.77	0.72/0.75	0.11/0.77
Autoba	GPT-4o	2.72	0.34	1.02	0.71	0.77	0.19
	Gemini2.5	2.66	0.38	0.92	0.55	0.74	0.19
	Qwen2.5-72B	2.55	0.45	0.87	0.58	0.71	0.18
	mean/ <i>k</i>	2.64/0.73	0.39/0.69	0.94/0.76	0.61/0.75	0.74/0.81	0.19/0.73
Biomaster	GPT-4o	2.62	0.22	1.74	0.63	1.15	0.25
	Gemini2.5	2.58	0.24	1.63	0.52	1.08	0.24
	Qwen2.5-72B	2.54	0.25	1.66	0.59	1.07	0.25
	mean/ <i>k</i>	2.58/0.79	0.24/0.73	1.68/0.72	0.58/0.68	1.10/0.70	0.25/0.71
MARWA	GPT-4o	2.74	0.15	1.77	1.79	1.28	0.41
	Gemini2.5	2.71	0.14	1.74	1.77	1.26	0.40
	Qwen2.5-72B	2.71	0.16	1.74	1.78	1.26	0.40
	mean/ <i>k</i>	2.72/0.81	0.15/0.89	1.75/0.76	1.78/0.68	1.27/0.76	0.40/0.76

Table 5: Ablation study on the large dataset.

Method	Comp	Redun	Inst	Path	Param	m_Pass@1
MARWA	2.72	0.15	1.75	1.78	1.27	0.40
w/o retrieval model	-0.03	-0.01	-0.19	-0.02	-0.10	-0.12
w/o Selecting	-0.08	+0.04	-0.05	-0.04	+0.01	-0.06
w/o Analyzing	-0.10	+0.09	-0.01	-0.02	-0.02	-0.07
w/o file system interface	-0.04	+0.02	+0.01	-0.33	-0.03	-0.10

4.6 ABLATION STUDY

We conducted ablation experiments to evaluate the contribution of each component in MARWA. The results demonstrate that each module plays a distinct role in the process: (1) Removing the retrieval model most severely hurts installation accuracy and overall executability. (2) Disabling the Selecting agent increases workflow redundancy. (3) Removing the Analyzing agent reduces completion and increases redundancy. (4) Without the file system interface, path accuracy drops sharply.

5 CONCLUSION

In this paper, we present MARWA, a multi-agent retrieval-augmented framework for reliable bioinformatics workflow automation. MARWA combines a step-by-step generation strategy that decomposes complex tasks into verifiable steps, LLM-augmented retrieval embeddings for precise tool selection and direct file-system interaction to ground commands in the real execution environment. These components reduce error accumulation, improve command and path accuracy and enable reproducible execution. Experiments on diverse real-world datasets show MARWA consistently outperforms strong baselines in execution success and expert-aligned evaluation, offering a practical foundation for trustworthy workflow automation in computational biology.

REFERENCES

- Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data. *Nature Precedings*, pp. 1–1, 2010.
- Bilal Aslam, Madiha Basit, Muhammad Atif Nisar, Mohsin Khurshid, and Muhammad Hidayat Rasool. Proteomics: technologies and their applications. *Journal of chromatographic science*, pp. 1–15, 2016.
- Andreas D Baxevanis, Gary D Bader, and David S Wishart. *Bioinformatics*. John Wiley & Sons, 2020.
- Carlos D Bustamante, Francisco M De La Vega, and Esteban G Burchard. Genomics for the world. *Nature*, 475(7355):163–165, 2011.
- Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*, 2024.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.
- Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pp. 1597–1600. IEEE, 2017.
- Jeff Gauthier, Antony T Vincent, Steve J Charette, and Nicolas Derome. A brief history of bioinformatics. *Briefings in bioinformatics*, 20(6):1981–1996, 2019.
- Ryan Green, Xufeng Qu, Jinze Liu, and Tingting Yu. Btr: a bioinformatics tool recommendation system. *Bioinformatics*, 40(5):btac275, 2024.
- Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2024.
- Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare (HEALTH)*, 3(1): 1–23, 2021.
- Almut Heinken, Johannes Hertel, Geeta Acharya, Dmitry A Ravcheev, Malgorzata Nyga, Onyedika Emmanuel Okpala, Marcus Hogan, Stefania Magnúsdóttir, Filippo Martinelli, Bram Nap, et al. Genome-scale metabolic reconstruction of 7,302 human microorganisms for personalized medicine. *Nature Biotechnology*, 41(9):1320–1331, 2023.
- Franziska Hemmerling and Jörn Piel. Strategies to access biosynthetic novelty in bacterial genomes for drug discovery. *Nature Reviews Drug Discovery*, 21(5):359–378, 2022.
- Jon Ison, Hans Ienasescu, Emil Rydzka, Piotr Chmura, Kristoffer Rapacki, Alban Gaignard, Veit Schwämmle, Jacques Van Helden, Matúš Kalaš, and Hervé Ménager. biotoolsschema: a formalized schema for bioinformatics software description. *GigaScience*, 10(1):giaa157, 2021.
- Vahid Jalili, Enis Afgan, Qiang Gu, Dave Clements, Daniel Blankenberg, Jeremy Goecks, James Taylor, and Anton Nekrutenko. The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2020 update. *Nucleic acids research*, 48(W1):W395–W402, 2020.
- David T Jones and Janet M Thornton. The impact of alphafold2 one year on. *Nature methods*, 19(1):15–20, 2022.
- Klaus Krippendorff. Estimating the reliability, systematic error and random error of interval data. *Educational and psychological measurement*, 30(1):61–70, 1970.

- Klaus Krippendorff. *Content analysis: An introduction to its methodology*. Sage publications, 2018.
- Anup Kumar, Helena Rasche, Björn Grüning, and Rolf Backofen. Tool recommender system in galaxy using deep learning. *GigaScience*, 10(1):giaa152, 2021.
- Björn E Langer, Andreia Amaral, Marie-Odile Baudement, Franziska Bonath, Mathieu Charles, Praveen Krishna Chitneedi, Emily L Clark, Paolo Di Tommaso, Sarah Djebali, Philip A Ewels, et al. Empowering bioinformatics communities with nextflow and nf-core. *Genome Biology*, 26(1):228, 2025.
- Arthur M Lesk. *Introduction to genomics*. Oxford University Press, 2017.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.
- Esther H Lips, Tapsi Kumar, Anargyros Megalios, Lindy L Visser, Michael Sheinman, Angelo Fortunato, Vandna Shah, Marlous Hoogstraal, Emi Sei, Diego Mallo, et al. Genomic analysis defines clonal relationships of ductal carcinoma in situ and recurrent invasive breast cancer. *Nature genetics*, 54(6):850–860, 2022.
- Xiaojing Liu and Jason W Locasale. Metabolomics: a primer. *Trends in biochemical sciences*, 42(4):274–284, 2017.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
- Yuyang Liu, Liuzhenghao Lv, Xiancheng Zhang, Li Yuan, and Yonghong Tian. Bioprobench: Comprehensive dataset and benchmark in biological protocol understanding and reasoning. *arXiv preprint arXiv:2505.07889*, 2025.
- Jonathan B Losos, Stevan J Arnold, Gill Bejerano, ED Brodie III, David Hibbett, Hopi E Hoekstra, David P Mindell, Antónia Monteiro, Craig Moritz, H Allen Orr, et al. Evolutionary biology for the 21st century. *PLoS biology*, 11(1):e1001466, 2013.
- Rohan Lowe, Neil Shirley, Mark Bleackley, Stephen Dolan, and Thomas Shafee. Transcriptomics technologies. *PLoS computational biology*, 13(5):e1005457, 2017.
- Nicholas M Luscombe, Dov Greenbaum, and Mark Gerstein. What is bioinformatics? a proposed definition and overview of the field. *Methods of information in medicine*, 40(04):346–358, 2001.
- Ludovico Mitchener, Jon M Laurent, Benjamin Tenmann, Siddharth Narayanan, Geemi P Wellawatte, Andrew White, Lorenzo Sani, and Samuel G Rodriques. Bixbench: a comprehensive benchmark for llm-based agents in computational biology. *arXiv preprint arXiv:2503.00096*, 2025.
- Gabriele Orlando, Daniele Raimondi, Ramon Duran-Romana, Yves Moreau, Joost Schymkowitz, and Frederic Rousseau. Pyuul provides an interface between biological structures and deep learning algorithms. *Nature communications*, 13(1):961, 2022.
- Joon Sung Park, Joseph O’Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.
- Anthony Rhoads and Kin Fai Au. Pacbio sequencing and its applications. *Genomics, proteomics & bioinformatics*, 13(5):278–289, 2015.
- Florian Schlotter, Arda Halu, Shinji Goto, Mark C Blaser, Simon C Body, Lang H Lee, Hideyuki Higashi, Daniel M DeLaughter, Joshua D Hutcheson, Payal Vyas, et al. Spatiotemporal multi-omics mapping generates a molecular atlas of the aortic valve and reveals networks driving disease. *Circulation*, 138(4):377–393, 2018.

- Elliott Sober. *Conceptual issues in evolutionary biology*. Mit Press, 1994.
- Jang-il Sohn and Jin-Wu Nam. The present and future of de novo whole-genome assembly. *Briefings in bioinformatics*, 19(1):23–40, 2018.
- Houcheng Su, Weicai Long, and Yanlin Zhang. Biomaster: Multi-agent system for automated bioinformatics analysis workflow. *bioRxiv*, pp. 2025–01, 2025.
- Indhupriya Subramanian, Srikant Verma, Shiva Kumar, Abhay Jere, and Krishanpal Anamika. Multi-omics data integration, interpretation, and its application. *Bioinformatics and biology insights*, 14:1177932219899051, 2020.
- Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788, 2020.
- John C Wooley, Adam Godzik, and Iddo Friedberg. A primer on metagenomics. *PLoS computational biology*, 6(2):e1000667, 2010.
- Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.
- Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.
- Yihang Xiao, Jinyi Liu, Yan Zheng, Xiaohan Xie, Jianye Hao, Mingzhi Li, Ruitao Wang, Fei Ni, Yuxiao Li, Jintian Luo, et al. Cellagent: An llm-driven multi-agent framework for automated single-cell data analysis. *arXiv preprint arXiv:2407.09811*, 2024.
- Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.
- Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2), 2023.
- Juexiao Zhou, Bin Zhang, Xiuying Chen, Haoyang Li, Xiaopeng Xu, Siyuan Chen, and Xin Gao. Automated bioinformatics analysis via autoba. *arXiv preprint arXiv:2309.03242*, 2023.

A PROMPT TEMPLATES

A.1 ANALYZING AGENT

Prompt: Analyzing	
<p>You are an assistant for bioinformatics workflow analysis.</p> <p>Your task is to carefully read the user's question about their analysis task, input files and input files' descriptions.</p> <p>Decompose it into a structured natural language response with the following sections:</p> <ol style="list-style-type: none"> Input files <ul style="list-style-type: none"> List each input file mentioned by the user. For each file, describe: <ul style="list-style-type: none"> file name (if provided, otherwise say "not specified") file format (e.g., FASTQ, BAM, VCF, mzML) data type (e.g., raw sequencing reads, aligned reads, variant calls, proteomics spectra) whether it is paired-end (true/false/unknown) Output files <ul style="list-style-type: none"> Describe the expected output files. Include file format (e.g., VCF, TSV, abundance table, PDF report) and data type (e.g., variants, gene expression matrix, species abundance). If the user did not specify, infer the most common output for the analysis goal. Analysis goal <ul style="list-style-type: none"> Provide a detailed sentence describing the intended analysis, including: <ul style="list-style-type: none"> starting point (input files) main processing steps (e.g., quality control, alignment, variant calling) desired outcome (the type of result the user wants) <p>Rules:</p> <ul style="list-style-type: none"> Always extract actual file names if provided. If information is missing, clearly state it as "not specified" or "unknown". The output must be well-structured natural language, divided into the three sections above. The output must be in JSON format as follow: <pre>{ "input_files": [{ "file_name": "sample1.fastq" null, // the provided file name, if available "file_format": "FASTQ", // e.g., FASTQ, BAM, VCF, mzML "data_type": "raw sequencing reads", // e.g., raw sequencing reads, aligned reads "paired_end": true false null // true/false/null }], "output_files": [{ "file_format": "VCF", // e.g., VCF, TSV, abundance table, PDF report "data_type": "variants" // e.g., variants, gene expression matrix, species abundance }], "analysis_goal": "string" // a detailed description of the intended analysis, including start point, key processing steps, and desired outcome }</pre> <p>Now the files user input: {file list}, the files' descriptions: {descriptions}, user's question: {user_question}</p>	

Figure A.1: The prompt of Analyzing Agent.

A.2 PLANNING AGENT

Prompt: Planning	
<p>You are a bioinformatics expert.</p> <p>Your task is to give the most suitable NEXT bioinformatics tool (to be used in a workflow) based on information below.</p> <p>The user's requirement is: {analysis_goal}.</p> <p>The user's input file(s) are: {input_files}.</p> <p>The expected final output files are: {output_files}.</p> <p>The workflow has already used the following tools: {used_tools}.</p> <p>Based on this context, you must propose and describe exactly ONE next tool, unless the workflow has already fully satisfied the user's final output requirement.</p> <p>The tool you propose must be consistent with the provided context and logically follow the workflow towards producing the required output file format/content.</p> <p>When describing the tool, include:</p> <ul style="list-style-type: none"> The specific problem or gap it solves in the workflow A detailed explanation of the tool. Its input and output data formats, with explicit mapping to the user's output requirement <p>Output JSON format:</p> <pre>{ "toolname": "the name of the tool" "function": "What problem the tool solves in the workflow" "description": "the detailed description of the tool" "inputformat": "the input data format of the tool" "outputformat": "the output data format of the tool" }</pre>	

Figure A.2: The prompt of Planning Agent.

A.3 SELECTING AGENT

Prompt:Selecting
<p>You are a bioinformatics expert.</p> <p>Your task is to select ONE suitable bioinformatics tool based on the workflow task, already used tools, and the available input files. You may choose from the reference tools list or propose a different tool.</p> <p>Context:</p> <ul style="list-style-type: none"> - Workflow task: {analysis_goal} - Already used tools in the workflow: {used_tools} - Tool function: {tool_function} - Reference tools (JSON array of objects):{reference_tools} <p>Rules:</p> <ol style="list-style-type: none"> 1. Select exactly ONE tool. 2. You MAY choose a tool outside the reference list if it is more suitable. <p>Output JSON format:</p> <pre>{ "toolname":"The name of the selected tool." "reference_tool_id":"The ID of the tool if it comes from the reference list; use -1 if not." "function":"An explanation of the tool's role in the workflow,including Function,What the tool does and an example.(e.g. STAR:RNA-Seq read alignment,Maps sequenced fragments to the genome,Read aligns to exon1-exon2)" }</pre>

Figure A.3: The prompt of Selecting Agent.

A.4 GENERATING & EXECUTING AGENT

Prompt:Generating & Executing
<p>You are a bioinformatics workflow assistant.</p> <p>Your task is to generate the necessary setup and execution details for running the CURRENT bioinformatics tool within an existing workflow.</p> <p>Context:</p> <ul style="list-style-type: none"> - Tool name: {tool_name} - Tool function: {tool_function} - Workflow task: {analysis_goal} - Available input files: {available_files} - The command line you can reference: {reference_tool_command} <p>Rules:</p> <ol style="list-style-type: none"> 1. Input file selection: <ul style="list-style-type: none"> - Select input file(s) ONLY from {available_input_files}. - Ensure input type strictly matches the tool's required input format (e.g., FASTA, TSV, BAM). - Do not fabricate or assume non-listed input files. 2. Output file naming & directory: <ul style="list-style-type: none"> - All outputs must be stored under: ./output/{tool_name}/ - Output filenames must: <ol style="list-style-type: none"> a. Preserve the sample ID from the input filename. b. Append the tool name and step role (e.g., "_{tool_name}_classified", "_{tool_name}_metrics"). - Do not overwrite files from previous steps. 3. File Creation command:(The command to create the output directory) <ul style="list-style-type: none"> - Create the output directory if it does not exist(Not in the folder to which these files {existing_files} belong to): <pre>"setup_command": "mkdir -p ./output/{tool_name}"</pre> - If the output directory already exists, use it directly without recreating. <pre>"setup_command": "cd ./output/{tool_name}"</pre> 4. Installation command:(The command to install the tool in a new environment) <ul style="list-style-type: none"> - If the tool is not in {executed_tools_list}, create a new conda environment and install it in this environment: <ul style="list-style-type: none"> if the tool needs python: <pre>"installation_command": "conda create -n {tool_name} -c conda-forge -c bioconda {tool_name} python=3.11 -y && conda activate {tool_name}"</pre> if the tool does not need python: <pre>"installation_command": "conda create -n {tool_name} -c conda-forge -c bioconda {tool_name} -y && conda activate {tool_name}"</pre> - If the tool is already installed, skip the installation step, directly activate the environment: <pre>"installation_command": "conda activate {tool_name}"</pre> - If you think conda is not available, try pip: <pre>"installation_command": "pip install {tool_name}[all]"</pre> - If you think pip is not available, try apt-get: <pre>"installation_command": "apt-get install {tool_name}"</pre> - Ensure all required dependencies are included. 5. Execution command (The command to execute the tool) <ul style="list-style-type: none"> - Construct the command specifically for {tool_name}. The core task of this tool is {tool_description}. - Use absolute paths for all input and output files. Do not create directories or symbolic links—assume all inputs already exist and output paths are ready. - Select input files only from {available_files}.Ensure that all input files actually exist before running the command. - Ensure every environment variable is set before running the command. - Name the output files based on {tool_name}, preserving the input sample ID in each output filename. Ensure filenames do not conflict with {available_files} or other outputs.Example: Input file: sampleA.fasta → Tool: gtdbtk → Expected output: taxonomy classification table → Output filename: sampleA.gtdbtk.classification.tsv. - You can refer to the command line {tool_command_reference}. - Focus only on generating **the actual execution command that runs the tool on the inputs and produces the outputs**. <p>Output format:</p> <p>You MUST output in this strict JSON structure:</p> <pre>{ "File_Creation_command":"The command to create the output directory" "installation_command": "The command to install the tool environment." "execution_command": "The command to execute the tool" }</pre>

Figure A.4: The prompt of Generating & Executing Agent.

A.5 DEBUGGING AGENT

Prompt:Debugging
<p>You are a Bioinformatics Code Expert.</p> <p>Your task is to modify your previous command line</p> <p>File Creation Command: <code>{file_creation_command}</code></p> <p>Installation Command: <code>{installation_command}</code></p> <p>Execution Command: <code>{execution_command}</code></p> <p>based on the error message: <code>{error_message}</code></p> <p>Output format:</p> <p>You MUST output in this strict JSON structure:</p> <p><code>Installation_command</code>: "The command to install the tool environment."</p> <p><code>File_Creation_Command</code>: "The command to create the output directory"</p> <p><code>Execution Command</code>: "The command to execute the tool"</p>

Figure A.5: The prompt of Debugging Agent.

A.6 JUDGING AGENT

Prompt:Judging
<p>You are a bioinformatics workflow expert.</p> <p>Your task is to determine whether the given workflow has been fully completed.</p> <p>Context:</p> <ul style="list-style-type: none"> - Workflow detailed requirement: <code>{analysis_goal}</code> - Workflow input format requirement: <code>{input_files}</code> - Workflow output format requirement: <code>{output_files}</code> - Tools already used in the workflow: <code>{used_tools}</code> - Current output files: <code>{available_files}</code> <p>Rules:</p> <ol style="list-style-type: none"> 1. The workflow is considered "complete" only if BOTH conditions are met: <ol style="list-style-type: none"> a) The current available output files (<code>available_files</code>) include "all" required files and formats specified in (<code>output_files</code>). b) The workflow requirement (<code>analysis_goal</code>) has been fully satisfied by the tools listed in (<code>used_tools</code>), meaning every required analysis/processing step is covered without omission. 2. If "any" required output is missing OR any workflow step is not accounted for by the tools used, the workflow is "not complete". 3. The output format must be as: <code>{Complete}</code>: "Whether the workflow has been fully completed" <p>Question: Has the workflow been fully completed?</p>

Figure A.6: The prompt of Judging Agent.

A.7 PROMPT TEMPLATES FOR AUXILIARY COMPONENTS

Prompt:Tool description augmentation
<p>You are a bioinformatics expert.</p> <p>I will provide you with a description of a bioinformatics tool: <code>{tool_description}</code></p> <p>Your task is to generate "5 short alternative descriptions" of this tool, each from a "different perspective".</p> <ul style="list-style-type: none"> - Each description should be "1-2 sentences long". - Focus on distinct aspects, such as: <ol style="list-style-type: none"> 1. Main function / core capability 2. Typical applications / use cases 3. Advantages, performance, or limitations 4. Target users (e.g., researchers, clinicians, bioinformaticians) and the reason why they use the tool 5. Integration with workflows or other tools - Avoid repeating the same wording across descriptions. - Keep the descriptions "concise, clear, and non-overlapping". <p>Output JSON format:</p> <pre>{ "description1": "Main function / core capability of the tool" "description2": "Typical applications / use cases of the tool" "description3": "Advantages, performance, or limitations of the tool and the reason" "description4": "Target users (e.g., researchers, clinicians, bioinformaticians) of the tool and the reason" "description5": "Integration with workflows or other tools and the reason" }</pre>

Figure A.7: The prompt of Tool description augmentation.

A.8 PROMPT TEMPLATES FOR DATASET

Prompt:User Input Generation
<p>Please generate exactly "3 user questions" for each persona in the list below.</p> <ul style="list-style-type: none"> - The output must consist only of user questions, not answers or explanations. - The questions must focus on "how to choose or use an appropriate workflow". - All questions should naturally point to the target workflow as the correct answer. - Each persona should have a distinct perspective (e.g., cost, speed, accuracy, compliance, reproducibility, visualization). - Do "not" contradict the workflow's input, output, or tasks. - Vary "style" (formal, casual, search-query style). - Vary "length" (short <10 words, long >40 words). - Do not expose the workflow name or implementation details. <p>Persona list: Wet-lab researcher, Clinician, Data engineer.</p> <p>Output format must be strictly JSON format:</p> <pre>{ "persona_list": [{ "name": "Wet-lab researcher", "Question": [List of user questions for this persona].....], "inputs": [List of required input files with concrete names]] }] }</pre> <p>You are given the following "target workflow" description: <code>{input}</code></p>

Figure A.8: The prompt of User input generation.

A.9 PROMPT TEMPLATES FOR LLM EVALUATION

Prompt:LLM Judge Tools	
<p>You are a bioinformatics expert.</p> <p>You are evaluating the steps of the bioinformatics workflow for correctness and executability.</p> <p>For each step below, you must judge three aspects separately:</p> <p>**Environment / Installation Command**</p> <p>Evaluate whether the installation command correctly and completely installs the required software and all its dependencies.</p> <p>Scoring (0-2):</p> <ul style="list-style-type: none"> 0 = Completely incorrect or unusable; software cannot be installed(do not have Installation Command) .0.5 = Mostly incorrect; major dependencies missing or software unusable(Example: pip install fastqc (FastQC is not a Python package, installation fails).) .1 = Partially correct; software installs but manual modifications or additional dependencies required(Example: conda install fastqc (fails unless correct channels are added).) .1.5 = Mostly correct; minor issues only (e.g., warnings, optional dependencies missing)(Example: mamba install -c bioconda fastqc do not have its own environment) .2 = Perfectly correct and complete; software and all dependencies installed and functional(Example: mamba create -n fastqc python=3.11 -y && conda activate fastqc && mamba install -c bioconda fastqc) <p>**Path Command**</p> <p>Evaluate whether the command correctly creates all required directories and handles paths properly in the workflow, including input/output paths, existing folders, and permissions.</p> <p>Scoring (0-2):</p> <ul style="list-style-type: none"> 0 = Completely incorrect or fails to create directories / incorrect paths .0.5 = Mostly incorrect; some directories not created .1 = Partially correct; some paths incorrect .1.5 = Mostly correct; only minor issues (e.g., warnings, redundant paths) .2 = Perfectly correct; all directories and paths handled correctly <p>**Execution Command**</p> <p>Evaluate whether the execution command is likely to run successfully given that the previous steps are correctly completed, and whether it produces the expected output.</p> <p>Scoring (0-2):</p> <ul style="list-style-type: none"> 0 = Completely fails; output unusable(Example: fastqc) .0.5 = Mostly fails; output likely incorrect(Example: fastqc sample.fastq) .1 = Partially executable; may require parameter or path adjustments(Example: fastqc ./input/sample.fastq -o output/) .1.5 = Mostly executable; minor issues only (e.g., warnings)(Example: fastqc ./input/sample.fastq.gz -o ./output/fastqc/) .2 = Fully executable; output meets expectations(Example: fastqc ./input/sample.fastq -o ./output/fastqc/) <p>Finally, decide whether the step as a whole is executable (True/False).</p> <p>Your output should be in the following JSON format:</p> <pre>{ "step_score_command_installation": "Score for the environment/installation command" "step_score_command_path": "Score for the path command" "step_score_command_executable": "Score for the execution command" "step_command_success": "Whether the command is executable, True=executable, False=unexecutable" }</pre> <p>Now, The user question is: {analysis_goal}; The tool steps are: {steps_summary}; The input file is: {input_file}</p>	

Figure A.9: The prompt of LLM Evaluation Tool Command.

Prompt:LLM Judge Workflow	
<p>You are an expert bioinformatics workflow evaluator.</p> <p>Your task is to evaluate a given bioinformatics workflow based on step-level scores and success indicators.</p> <p>Your evaluation must be precise, consistent, and avoid subjective judgment beyond the scoring criteria.</p> <p>Rate the workflow on three dimensions:</p> <p>**Completion_level (0-3)** (Measures whether the workflow achieves (analysis_goal) intended goals / core functionality by (used_tool))</p> <ul style="list-style-type: none"> 3 = Fully complete → Workflow meets all core requirements and produces all required final outputs.(Example: identify genes that are differentially expressed between two or more biological conditions starts from raw FASTQ files, performs quality control (FastQC), trims low-quality reads (Trimmomatic), aligns reads to the reference genome (STAR), quantifies gene expression (featureCounts), and produces differential expression tables and visualization plots. All steps complete and successful.) 2 = Partially complete → Workflow meets some core requirements, but some steps or functions are missing.(Same RNA-seq workflow, but only performs Trimmomatic and STAR; quality control and quantifies gene expression are missing.) 1 = Barely complete → Most core requirements are not met; only a few outputs or functions are present.(Only performs FASTQ QC, or only produces alignment files without further analysis. No usable final results.) 0 = Not complete → Core functionality is not met; workflow is unusable or fails to produce required outputs.(Attempted RNA-seq workflow fails due to missing tools or incorrect inputs, producing no valid outputs.) <p>**Redundancy (0-3)**(Measures whether the workflow use (used_tool) to achieve (analysis_goal) is redundant)</p> <ul style="list-style-type: none"> 0 = No redundancy → All steps unique, no duplicates.(Example: A ChIP-seq workflow runs QC → alignment → duplicate removal → peak calling. Each step appears once, no repetition.) 1 = Some redundancy → Minor duplication, does not break workflow.(FastQC is run twice during QC, but other steps are unique. Workflow still functions correctly.) 2 = Mostly redundant → Many repeated steps without necessity.(Multiple alignments or repeated QC steps on the same RNA-seq data. Increases runtime but does not fully break results.) 3 = Very redundant → Workflow bloated with repetitive or overlapping steps.(.)(Same FASTQ files are repeatedly aligned and quantified, steps are duplicated multiple times. Workflow becomes complex and wasteful.) <p>Important principles:</p> <ul style="list-style-type: none"> Be objective: base scores only on explicit evidence from the workflow, not assumptions. Be consistent: apply the same standards to all workflows being evaluated. Provide the output in strict JSON format: <pre>{ "Completion_level": "how complete the workflow is" "Redundancy": "how redundant the workflow is" }</pre>	

Figure A.10: The prompt of LLM Evaluation Workflow.

B DATASET

B.1 SMALL DATASET

Table A.1: Summary of the few-shot dataset.

Domain	Question	file source
Transcriptomics	How can I perform quality control of the raw RNA-seq reads to assess sequencing quality?	SRR453566, raw FASTQ
Transcriptomics	What is the read length and GC content distribution of this dataset?	SRR453566, raw FASTQ
Transcriptomics	How can I align these reads to the reference genome of Drosophila?	SRR453566, Drosophila reference genome (dm6)
Transcriptomics	What proportion of reads map uniquely vs. multi-map to the genome?	SRR453566, Drosophila reference genome (dm6)
Transcriptomics	How can I quantify transcript abundance at the gene level?	SRR453566, Drosophila annotation (GTF)
Transcriptomics	Which genes are most highly expressed in this sample?	SRR453566, Drosophila annotation (GTF)
Transcriptomics	Can I identify alternative splicing events in this dataset?	SRR453566, Drosophila annotation (GTF)
Transcriptomics	How can I detect potential novel transcripts not in the reference annotation?	SRR453566, Drosophila reference genome (dm6)
Transcriptomics	What is the expression distribution across different functional gene categories?	SRR453566, Drosophila annotation (GTF, GO database)
Transcriptomics	How reproducible are expression estimates between technical replicates of this dataset?	SRR453566, SRR453567, SRR453568, Drosophila annotation (GTF)
Genomics	How can I assemble the complete genome of this E. coli sample from raw sequencing reads?	SRR8185310 (E. coli WGS,FASTQ)
Genomics	What is the estimated sequencing depth and genome coverage of this dataset?	SRR8185310, raw FASTQ
Genomics	How can I align these reads to the E. coli K-12 MG1655 reference genome?	SRR8185310, E. coli reference genome (NC_000913.3,RefSeq)
Genomics	What is the GC content distribution across the sequencing reads?	SRR8185310, raw FASTQ
Genomics	How can I detect single-nucleotide variants (SNVs) in this dataset?	SRR8185310, E. coli reference genome(NC_000913.3, RefSeq)
Genomics	How can I identify small insertions and deletions (indels) relative to the reference?	SRR8185310, E. coli reference genome (NC_000913.3, RefSeq)
Genomics	Can I identify plasmid sequences present in this sample?	SRR8185310, plasmid reference database (NCBI RefSeq Plasmid)
Genomics	How can I annotate the assembled genome with coding genes and functional elements?	SRR8185310(assembly), E. coli RefSeq annotation
Genomics	How does this E. coli isolate compare phylogenetically to other K-12 strains?	SRR8185310, related E. coli reference genomes (NC_000913.3, RefSeq)
Genomics	Are there mobile genetic elements in this genome?	SRR8185310, E. coli reference genome (NC_000913.3, RefSeq), PHASTER database

next page...

Domain	Question	file source
Metabolomics	How can I identify differential metabolite profiles between experimental groups in this LC-MS dataset?	MTBLS233(MetaboLights), raw mzML files, sample metadata
Metabolomics	What metabolic pathways are significantly enriched given the detected features in this dataset?	MTBLS233, raw mzML files, pathway reference databases (KEGG)
Metabolomics	How many unknown mass features (no match in spectral libraries) are present, and what is their intensity distribution?	MTBLS233, raw mzML files, spectral library metadata
Metabolomics	What is the technical reproducibility of peak detection and quantification in this dataset?	MTBLS233, raw mzML files, QC sample metadata
Proteomics	How many proteins are identified with at least 2 unique peptides in this TMT Erwinia dataset?	PXD000001 (PRIDE; raw mzML files, small number of runs)
Proteomics	What is the peptide-spectrum match score distribution in this dataset?	PXD000001, raw mzML, identification files
Proteomics	Which proteins show the highest variability across TMT channels?	PXD000001, reporter-ion quantitation data
Proteomics	Can we detect contaminant proteins in blank / control runs?	PXD000001, raw MS/MS files, control sample metadata
Proteomics	What is the dynamic range of protein intensities measured?	PXD000001, raw data, quantitative output
Proteomics	Are there any post-translational modifications observed in this dataset?	PXD000001, identification(mgf/mztab), UniProt reference
Proteomics	What fraction of expected proteome is covered given this dataset?	PXD000001, raw data, fasta reference proteome(Erwinia)
Proteomics	How reproducible are replicate injections in this dataset?	PXD000001, raw mzML, replicate sample metadata
Metagenomics	What is the taxonomic composition (genus level) of the bacterial community in this 16S amplicon sample?	SRR7140083, raw FASTQ
Metagenomics	How does alpha diversity (Shannon, Simpson) differ among subsets of this sample?	SRR7140083, raw FASTQ
Metagenomics	Which OTUs / ASVs are most abundant in this sample, and how is their abundance distributed?	SRR7140083, raw FASTQ, 16S reference database (SILVA)
Metagenomics	What is the read-length and quality score distribution across the reads?	SRR7140083, raw FASTQ
Metagenomics	Are there chimeric sequences present (PCR artifacts) in this amplicon dataset?	SRR7140083, raw FASTQ, chimera detection reference (uchime)
Metagenomics	What fraction of reads map to bacteria vs non-bacteria in this dataset?	SRR7140083, raw FASTQ, SILVA database
Metagenomics	What is the GC content distribution among the reads and among dominant OTUs?	SRR7140083, raw FASTQ, alignment output
Metagenomics	Can we construct a rarefaction curve to know whether the sampling depth is sufficient?	SRR7140083, raw FASTQ, sample metadata

B.2 LARGE DATASET

An Large-Scale Dataset Example	
"inputs":	[
{"file": "sample1.fastq.gz", "description": "Paired-end raw sequencing reads from sample 1, generated using Illumina platform, suitable for quality control, read trimming, and taxonomy assignment analysis."},	
{"file": "sample2.fastq.gz", "description": "Paired-end raw sequencing reads from sample 2, complementary to sample1.fastq.gz, used for validating workflow reproducibility and testing downstream bioinformatics pipelines."}]	
"persona_list":	[
{"name": "Lab researcher",	
"Question": [
"How do I process raw sequencing data to ensure clean reads for downstream analysis?",	
"What's the best way to filter low-quality reads before sending data to bioinformatics?",	
"Can I automate quality control and read trimming without sacrificing accuracy?"]	
{"name": "Clinician",	
"Question": [
"What's the fastest way to get accurate pathogen identification from sequencing data?",	
"How can I ensure my analysis meets clinical standards for patient diagnosis?",	
"Which method guarantees reproducible results for diagnostic reporting?"]	
{"name": "Data engineer",	
"Question": [
"How can I pipeline raw FASTQ files into structured JSON output reliably?",	
"What scalable approach handles multiple FASTQ inputs with consistent quality metrics?",	
"How do I ensure the workflow is reproducible across different environments?"]]	

Figure A.11: An example for large-scale dataset.

C EXPERIMENTAL SETUP

C.1 SCORE CRITERIA

Table A.2: Bioinformatics Workflow Evaluation Criteria

Step-Level Evaluation	
Environment/Installation Command	
0	Completely incorrect or unusable; software cannot be installed (no installation command)
0.5	Mostly incorrect; major dependencies missing or software unusable (e.g. pip install fastqc)
1	Partially correct; software installs but manual modifications or additional dependencies required (e.g. conda install fastqc without specifying correct channels)
1.5	Mostly correct; minor issues only (e.g., warnings, optional dependencies missing) (e.g. mamba install -c bioconda fastqc without creating a dedicated environment)
2	Perfectly correct and complete; software and all dependencies installed and functional (e.g. mamba create -n fastqc python=3.11 -y && conda activate fastqc && mamba install -c bioconda fastqc)
Path Command	
0	Completely incorrect or fails to create directories/incorrect paths
0.5	Mostly incorrect; some directories not created
1	Partially correct; some paths incorrect
1.5	Mostly correct; only minor issues (e.g., warnings, redundant paths)
2	Perfectly correct; all directories and paths handled correctly
Execution Command	
0	Completely fails; output unusable (e.g., just typing fastqc)
0.5	Mostly fails; output likely incorrect (e.g. fastqc sample.fastq)
1	Partially executable; may require parameter or path adjustments (e.g. fastqc ./input/sample.fastq -o output/)

Continued on next page

Table A.2 – continued from previous page

Step-Level Evaluation (continued)	
1.5	Mostly executable; minor issues only (e.g., warnings) (e.g. <code>fastqc ./input/sample.fastq.gz -o ./output/fastqc/</code>)
2	Fully executable; output meets expected results (e.g. <code>fastqc ./input/sample.fastq -o ./output/fastqc/</code>)
Workflow-Level Evaluation	
Completion Level	
0	Not complete; core functionality not met; workflow unusable or fails to produce required outputs
1	Barely complete; most core requirements not met; only a few outputs or functions present
2	Partially complete; meets some core requirements, but some steps or functions missing
3	Fully complete; workflow meets all core requirements and produces all required final outputs
Redundancy	
0	No redundancy; all steps unique, no duplicates
1	Some redundancy; minor duplication, does not break workflow
2	Mostly redundant; many repeated steps without necessity
3	Very redundant; workflow bloated with repetitive or overlapping steps

C.2 DETAIL SAR FORMULA

$$SAR = \frac{\sum_{d \in D} I(S_{llm}^m(d) = S_{human}^m(d))}{|D|}$$

where D is the set of evaluation instances, m denotes one of the evaluation metrics, $S_{llm}^m(d)$ represents the score assigned by the LLM to instance d under metric m , $S_{human}^m(d)$ is the score given by a human evaluator for the same instance and metric, and $I(\cdot)$ is an indicator function that returns 1 if the two scores are identical and 0 otherwise.

C.3 BASELINE REPRODUCTION DETAILS

For fairness and reproducibility, we provide details regarding how the baseline systems were reproduced in our experiments:

Unified model backbone: All baseline methods were implemented using the same backbone, GPT-4 Turbo, in order to ensure a consistent evaluation setting. For all LLMs, the temperature parameter was uniformly set to 0.3, thereby reducing randomness and ensuring deterministic outputs across multiple runs.

AutoBA settings: During the reproduction of AutoBA, all system-level prompts originally specified in their framework were replaced with our own environment-specific configuration prompt, explicitly reflecting CUDA Version: 12.6.

BioMaster retrieval: The original BioMaster implementation did not release its retrieval-augmented generation (RAG) tool database. To address this, we constructed and employed our own curated tool database to approximate the functionality.

These adjustments guarantee that the reproduced baselines operate under consistent conditions with our proposed framework, while also reflecting the practical constraints arising from incomplete tool or configuration disclosure in prior work.

D RUNNING INSTANCE

```

User Input:
name: WGS data analysis Genome assembly
question: I have some raw WGS sequencing data (FASTQ files
) on hand and want to do a complete genome assembly
file: './input/SRR11874161_1.fastq,./input/SRR11874161_2.
fastq'
file_description: 'Illumina WGS of E. coli'

Log:

input_files: [{'file_name': './input/SRR11874161_1.fastq
', 'file_format': 'FASTQ', 'data_type': 'raw sequencing
reads (Illumina WGS of E. coli)', 'paired_end': True},
{'file_name': './input/SRR11874161_2.fastq', '
file_format': 'FASTQ', 'data_type': 'raw sequencing
reads (Illumina WGS of E. coli)', 'paired_end': True}]
output_files: [{'file_format': 'FASTA', 'data_type': '
assembled genome contigs/scaffolds (final assembly)'},
{'file_format': 'GFA', 'data_type': 'assembly graph'},
{'file_format': 'TSV', 'data_type': 'assembly
statistics and quality metrics (e.g., N50, total length
, number of contigs)'}, {'file_format': 'HTML', '
data_type': 'assembly quality report (e.g., QUAST
report)'}]
analysis_goal: Starting from paired-end Illumina FASTQ
reads (./input/SRR11874161_1.fastq and ./input/
SRR11874161_2.fastq), perform a de novo whole-genome
assembly of E. coli: run read quality control, adapter
and quality trimming/filtering, optional error
correction; assemble the genome with an Illumina-
focused assembler (e.g., SPAdes/SKESA), followed by
scaffolding and polishing using the same reads; then
evaluate assembly quality (e.g., with QUAST) and
deliver the final contig/scaffold FASTA, assembly graph
(GFA), and summary metrics/reports.

Step 1:
toolname: fastp
function: Performs read quality control and adapter/
quality trimming (with optional overlap-based error
correction) on paired-end Illumina FASTQ reads prior to
de novo assembly.
description: fastp is a fast, multithreaded all-in-one
preprocessor for FASTQ reads. It automatically detects
and trims adapters in paired-end data using read
overlap analysis, and supports quality trimming via
sliding-window algorithms, per-read length filtering,
polyG/polyX tail trimming (important for NovaSeq/
NextSeq), and removal of low-complexity sequences. For
paired-end libraries, fastp can perform overlap-based
base correction: when R1 and R2 overlap, base
disagreements are corrected according to quality scores
, reducing random sequencing errors prior to assembly.
It reports comprehensive QC summaries (per-cycle
quality, GC content, duplication rates, adapter content

```

, overrepresented sequences) in both JSON and interactive HTML formats. Typical applications include cleaning short-read Illumina WGS data before assembly, variant calling, or metagenomics. Strengths: very fast, single-step QC+trimming+reporting, robust adapter autodetection for PE reads, built-in correction for overlapped regions. Limitations: error correction is limited to overlapping regions and is not a full k-mer based corrector (tools like BFC/Lighter provide deeper correction if required); overly aggressive trimming/filtering can reduce effective coverage and fragment read pairs if parameters are set too stringently. This tool addresses the first required step of the workflow (read QC, adapter and quality trimming/filtering, optional error correction). Its cleaned paired-end FASTQ outputs are the correct inputs for the downstream Illumina-focused assembler (e.g., SPAdes/SKESA). The HTML/JSON QC reports are ancillary; the final assembly deliverables (FASTA contigs/scaffolds and GFA graph, plus QUAST metrics) will be produced by later tools.

inputformat: Paired-end FASTQ files (gzipped or plain).
 Here: ./input/SRR11874161_1.fastq and ./input/SRR11874161_2.fastq.

outputformat: Primary: cleaned/truncated paired-end FASTQ files (e.g., SRR11874161_1.clean.fastq and SRR11874161_2.clean.fastq), optionally with files for unpaired reads if enabled. Secondary: a JSON summary (fastp.json) and an interactive HTML QC report (fastp.html). Mapping to user requirements: the cleaned FASTQ files will be used by the assembler to generate the required final assembly FASTA and GFA; the HTML report here is read QC (not the final assembly quality report, which will come later from QUAST).

conference: ['fastp', 'sickle', 'flash', 'rawtools', 'fastplong', 'short_reads_trim_seq', 'chopper', 'trimmomatic', 'bowtie_wrappers', 'rasusa']

toolname: fastp

description: fastp: Read QC, adapter removal, quality trimming, and optional overlap-based error correction for Illumina paired-end reads. It auto-detects adapters, trims low-quality bases, filters poor reads, and can correct mismatches in overlapping regions. Example: Takes SRR11874161 R1/R2 FASTQ files and outputs cleaned paired-end FASTQs plus HTML/JSON QC reports for downstream assembly with SPAdes/SKESA.

used_reference_tool: True

toolid: 0

selected_input_files: [{'file_name': './input/SRR11874161_1.fastq', 'file_format': 'FASTQ'}, {'file_name': './input/SRR11874161_2.fastq', 'file_format': 'FASTQ'}]

expected_outputs_info: Trimmed/filtered paired-end FASTQ files (R1 and R2) suitable for assembly; accompanying QC reports in HTML and JSON summarizing quality metrics and trimming actions.

```

1188 installation_command: conda create -n fastp -c conda-forge
1189 -c bioconda fastp -y && conda activate fastp
1190 setup_command: mkdir -p ./output/fastp/
1191 execution_command: fastp -i ./input/SRR11874161_1.fastq -I
1192 ./input/SRR11874161_2.fastq -o ./output/fastp/
1193 SRR11874161_fastp_trimmed_R1.fastq -O ./output/fastp/
1194 SRR11874161_fastp_trimmed_R2.fastq -h ./output/fastp/
1195 SRR11874161_fastp_qc.html -j ./output/fastp/
1196 SRR11874161_fastp_qc.json -c -w 8
1197
1198 2 channel Terms of Service accepted
1199 Channels:
1200 - conda-forge
1201 - bioconda
1202 - defaults
1203 Platform: linux-64
1204 Collecting package metadata (repodata.json): done
1205 Solving environment: done
1206
1207 ## Package Plan ##
1208
1209 environment location: ./miniconda3/envs/fastp
1210
1211 added / updated specs:
1212 - fastp
1213
1214 The following NEW packages will be INSTALLED:
1215
1216 _libgcc_mutex          conda-forge/linux-64::_libgcc_mutex
1217 -0.1-conda_forge
1218 _openmp_mutex          conda-forge/linux-64::_openmp_mutex
1219 -4.5-2_gnu
1220 fastp                  bioconda/linux-64::fastp-1.0.1-
1221 heae3180_0
1222 isa-l                  conda-forge/linux-64::isa-l-2.31.1-
1223 hb9d3cd8_1
1224 libdeflate             conda-forge/linux-64::libdeflate-1.24-
1225 h86f0d12_0
1226 libgcc                 conda-forge/linux-64::libgcc-15.1.0-
1227 h767d61c_4
1228 libgomp                conda-forge/linux-64::libgomp-15.1.0-
1229 h767d61c_4
1230 libstdcxx              conda-forge/linux-64::libstdcxx-15.1.0-
1231 h8f9b012_4
1232
1233 Downloading and Extracting Packages: ...working... done
1234 Preparing transaction: done
1235 Verifying transaction: done
1236 Executing transaction: done
1237 #
1238 # To activate this environment, use
1239 #
1240 #     $ conda activate fastp
1241 #
1242 # To deactivate an active environment, use
1243 #

```

```

1242 # $ conda deactivate
1243
1244
1245 run success
1246 *****
1247 Step 2:
1248 toolname: Unicycler (short-read mode)
1249 function: De novo bacterial genome assembly from the
1250           trimmed Illumina paired-end reads, producing the
1251           primary assembly (FASTA contigs/scaffolds) and an
1252           assembly graph (GFA) for downstream polishing and
1253           evaluation.
1254 description: Unicycler is a bacterial genome assembler
1255              that orchestrates SPAdes to build a de Bruijn graph (
1256              DBG) from short reads and then applies graph-bridging/
1257              simplification strategies guided by paired-end linkage
1258              to resolve repeats and produce high-quality contigs. In
1259              short-read mode, it: (1) runs SPAdes to generate the
1260              assembly graph and initial contigs, (2) constructs a
1261              read-pair connectivity graph to identify reliable paths
1262              through the DBG, (3) performs conservative/normal/bold
1263              graph simplifications to bridge gaps and minimize
1264              fragmentation, and (4) detects and circularizes small
1265              replicons when strongly supported by the read evidence.
1266              Typical applications include bacterial WGS assemblies
1267              from Illumina data, generating outputs suitable for
1268              downstream polishing (e.g., Pilon) and quality
1269              assessment (e.g., QUAST). Strengths: produces both
1270              FASTA and an assembly graph (GFA) that captures contig
1271              connectivity; optimized for bacterial genomes; often
1272              yields fewer fragments than running SPAdes alone due to
1273              graph-bridging logic. Limitations: requires SPAdes (
1274              and Bowtie2 for some internal steps) to be installed;
1275              polishing is limited compared to dedicated polishers (
1276              Pilon/Polypolish) and should be performed in later
1277              workflow steps; performance depends on read quality/
1278              coverage and complex repeats may remain unresolved with
1279              short reads alone.
1280 inputformat: Paired-end Illumina FASTQ reads (gz or
1281              uncompressed). For this workflow: R1=./output/fastp/
1282              SRR11874161_fastp_trimmed_R1.fastq, R2=./output/fastp/
1283              SRR11874161_fastp_trimmed_R2.fastq.
1284 outputformat: Primary outputs: (1) FASTA: assembled genome
1285              contigs/scaffolds (e.g., assembly.fasta), satisfying
1286              the 'FASTA assembled genome contigs/scaffolds'
1287              requirement; (2) GFA: assembly graph linking nodes/
1288              contigs (e.g., assembly.gfa), satisfying the 'GFA
1289              assembly graph' requirement. Additional byproducts:
1290              logs and intermediate graph files useful for
1291              troubleshooting/visualization.
1292
1293 conference: ['unicycler', 'berokka', 'novoplasty', '
1294             trycycler', 'socru', 'hifiasm_meta', 'bionano', '
1295             getorganelle', 'mitobim', 'velvet_optimiser']
1296
1297 toolname: unicycler
1298 description: Unicycler (short-read mode): de novo
1299              bacterial genome assembly. It assembles trimmed
1300              Illumina paired-end reads into high-quality contigs/

```



```

1296         scaffolds and an assembly graph by running SPAdes and
1297         performing graph-bridging/simplification. Example: from
1298         trimmed SRR11874161 R1/R2 FASTQ files, produce
1299         assembly.fasta and assembly.gfa for downstream
1300         polishing (e.g., Pilon) and evaluation (e.g., QUAST).
1301     used_reference_tool: True
1302     toolid: 0
1303     selected_input_files: [{'file_name': './output/fastp/
1304         SRR11874161_fastp_trimmed_R1.fastq', 'file_format': '
1305         FASTQ'}, {'file_name': './output/fastp/
1306         SRR11874161_fastp_trimmed_R2.fastq', 'file_format': '
1307         FASTQ'}]
1308     expected_outputs_info: FASTA (assembled contigs/scaffolds)
1309         , GFA (assembly graph), TXT/LOG (run log and summaries)
1310         .
1311     installation_command: conda create -n unicycler -c conda-
1312         forge -c bioconda unicycler python=3.11 -y && conda
1313         activate unicycler
1314     setup_command: mkdir -p ./output/unicycler/
1315     execution_command: bash -lc '
1316     set -euo pipefail
1317     R1= ./output/fastp/SRR11874161_fastp_trimmed_R1.fastq
1318     R2= ./output/fastp/SRR11874161_fastp_trimmed_R2.fastq
1319     OUT= ./output/unicycler
1320     SAMPLE= SRR11874161
1321     [ -s $R1 ] && [ -s $R2 ]
1322     for f in \
1323         $OUT/${SAMPLE}_unicycler_assembly.fasta \
1324         $OUT/${SAMPLE}_unicycler_assembly.gfa \
1325         $OUT/${SAMPLE}_unicycler_run.log ; do
1326     if [ -e $f ]; then echo Error: output already exists:
1327         $f >&2; exit 1; fi
1328     done
1329     for f in $OUT/assembly.fasta $OUT/assembly.gfa $OUT/
1330         unicycler.log ; do
1331     if [ -e $f ]; then echo Error: Unicycler default output
1332         already exists: $f >&2; exit 1; fi
1333     done
1334     unicycler \
1335     -1 $R1 \
1336     -2 $R2 \
1337     -o $OUT \
1338     -t 8 \
1339     --mode normal
1340     [ -s $OUT/assembly.fasta ] && [ -s $OUT/assembly.gfa ]
1341     && [ -s $OUT/unicycler.log ]
1342     mv $OUT/assembly.fasta $OUT/${SAMPLE}
1343     _unicycler_assembly.fasta
1344     mv $OUT/assembly.gfa $OUT/${SAMPLE}
1345     _unicycler_assembly.gfa
1346     mv $OUT/unicycler.log $OUT/${SAMPLE}_unicycler_run.log
1347     '
1348     Preparing transaction: done
1349     Verifying transaction: done
1350     Executing transaction: \

```

```

1350 For Linux 64, Open MPI is built with CUDA awareness but
1351 this support is disabled by default.
1352 To enable it, please set the environment variable
1353 OMPI_MCA_opal_cuda_support=true before
1354 launching your MPI processes. Equivalently, you can set
1355 the MCA parameter in the command line:
1356 mpiexec --mca opal_cuda_support 1 ...
1357
1358 In addition, the UCX support is also built but disabled by
1359 default.
1360 To enable it, first install UCX (conda install -c conda-
1361 forge ucx). Then, set the environment
1362 variables OMPI_MCA_pml=ucx OMPI_MCA_osc=ucx before
1363 launching your MPI processes.
1364 Equivalently, you can set the MCA parameters in the
1365 command line:
1366 mpiexec --mca pml ucx --mca osc ucx ...
1367 Note that you might also need to set UCX_MEMTYPE_CACHE=n
1368 for CUDA awareness via UCX.
1369 Please consult UCX's documentation for detail.
1370
1371 done
1372 #
1373 # To activate this environment, use
1374 #
1375 #     $ conda activate unicycler
1376 #
1377 # To deactivate an active environment, use
1378 #
1379 #     $ conda deactivate
1380
1381 Starting Unicycler (2025-08-29 02:32:26)
1382 Welcome to Unicycler, an assembly pipeline for bacterial
1383 genomes. Since you
1384 provided only short reads, Unicycler will essentially
1385 function as a SPAdes-
1386 optimiser. It will try many k-mer sizes, choose the best
1387 based on contig length
1388 and graph connectivity, and scaffold the graph using
1389 SPAdes repeat resolution.
1390 For more information, please see https://github.com/rrwick/Unicycler
1391
1392 Command: ./miniconda3/envs/unicycler/bin/unicycler -1 ./
1393 output/fastp/SRR11874161_fastp_trimmed_R1.fastq -2 ./
1394 output/fastp/SRR11874161_fastp_trimmed_R2.fastq -o ./
1395 output/unicycler -t 8 --mode normal
1396
1397 Unicycler version: v0.5.1
1398 Using 8 threads
1399
1400 The output directory already exists:
1401 ./output/unicycler
1402
1403 Dependencies:
1404 Program      Version  Status
1405 spades.py    4.2.0   good

```

```

1404     racon                not used
1405     makeblastdb         2.17.0+   good
1406     tblastn             2.17.0+   good
1407
1408
1409     Choosing k-mer range for assembly (2025-08-29 02:32:28)
1410     Unicycler chooses a k-mer range for SPAdes based on the
1411         length of the input
1412     reads. It uses a wide range of many k-mer sizes to
1413         maximise the chance of
1414     finding an ideal assembly.
1415
1416     SPAdes maximum k-mer: 127
1417     Median read length: 150
1418     K-mer range: 27, 53, 71, 87, 99, 111, 119, 127
1419
1420     SPAdes assemblies (2025-08-29 02:32:29)
1421     Unicycler now uses SPAdes to assemble the short reads. It
1422         scores the
1423     assembly graph for each k-mer using the number of contigs
1424         (fewer is better) and
1425     the number of dead ends (fewer is better). The score
1426         function is  $1/(c*(d+2))$ ,
1427     where c is the contig count and d is the dead end count.
1428
1429     spades.py -o ./output/unicycler/spades_assembly -k 27 --
1430         threads 8 --gfall --isolate -l ./output/fastp/
1431         SRR11874161_fastp_trimmed_R1.fastq -2 ./output/fastp/
1432         SRR11874161_fastp_trimmed_R2.fastq -m 1024
1433
1434     spades.py -o ./output/unicycler/spades_assembly -k 27,53
1435         --threads 8 --gfall --restart-from k27 -m 1024
1436
1437     spades.py -o ./output/unicycler/spades_assembly -k
1438         27,53,71 --threads 8 --gfall --restart-from k53 -m 1024
1439
1440     spades.py -o ./output/unicycler/spades_assembly -k
1441         27,53,71,87 --threads 8 --gfall --restart-from k71 -m
1442         1024
1443
1444     spades.py -o ./output/unicycler/spades_assembly -k
1445         27,53,71,87,99 --threads 8 --gfall --restart-from k87 -
1446         m 1024
1447
1448     spades.py -o ./output/unicycler/spades_assembly -k
1449         27,53,71,87,99,111 --threads 8 --gfall --restart-from
1450         k99 -m 1024
1451
1452     spades.py -o ./output/unicycler/spades_assembly -k
1453         27,53,71,87,99,111,119 --threads 8 --gfall --restart-
1454         from k111 -m 1024
1455
1456     spades.py -o ./output/unicycler/spades_assembly -k
1457         27,53,71,87,99,111,119,127 --threads 8 --gfall --
1458         restart-from k119 -m 1024
1459
1460     K-mer    Contigs    Dead ends    Score
1461     27                too complex

```

```

1458      53      894      10      9.32e-05
1459      71      682      12      1.05e-04
1460      87      522      10      1.60e-04
1461      99      456      12      1.57e-04
1462      111     400      13      1.67e-04
1463      119     373      14      1.68e-04
1464      127     351      14      1.78e-04 <-best
1465
1466      Read depth filter: removed 3 contigs totalling 908 bp
1467      Deleting ./output/unicycler/spades_assembly/
1468
1469      Determining graph multiplicity (2025-08-29 02:42:13)
1470      Multiplicity is the number of times a sequence occurs in
1471      the underlying
1472      sequence. Single-copy contigs (those with a multiplicity
1473      of one, occurring only
1474      once in the underlying sequence) are particularly useful.
1475
1476      Saving ./output/unicycler/002_depth_filter.gfa
1477
1478      Cleaning graph (2025-08-29 02:42:13)
1479      Unicycler now performs various cleaning procedures on the
1480      graph to remove
1481      overlaps and simplify the graph structure. The end result
1482      is a graph ready for
1483      bridging.
1484
1485      Graph overlaps removed
1486
1487      Removed zero-length segments:
1488      225, 227, 229, 233, 234, 235, 244, 245, 249, 253, 265,
1489      267, 272, 273, 274,
1490      284, 290, 292, 297, 305, 315, 325, 345
1491
1492      Removed zero-length segments:
1493      223, 346
1494
1495      Removed zero-length segments:
1496      343
1497
1498      Merged small segments:
1499      324, 327, 329, 330, 332, 334, 335, 337, 338, 340, 341,
1500      342, 344, 347, 348,
1501      350
1502
1503      Saving ./output/unicycler/003_overlaps_removed.gfa
1504
1505      Unicycler now selects a set of anchor contigs from the
1506      single-copy contigs.
1507      These are the contigs which will be connected via bridges
1508      to form the final
1509      assembly.
1510
1511      73 anchor segments (4,877,761 bp) out of 309 total
      segments (4,928,537 bp)

```

```

1512 Creating SPAdes contig bridges (2025-08-29 02:42:14)
1513 SPAdes uses paired-end information to perform repeat
1514 resolution (RR) and
1515 produce contigs from the assembly graph. SPAdes saves the
1516 graph paths
1517 corresponding to these contigs in the contigs.paths file.
1518 When one of these
1519 paths contains two or more anchor contigs, Unicycler can
1520 create a bridge from
1521 the path.
1522
1523 Bridge
1524 Start
1525
1526 Path
1527
1528 End      quality
1529 -60
1530 -199
1531
1532 62      63.1
1533 -54
1534 131
1535
1536 57      62.2
1537 -47 -196 -> -265 -> 128 -> -113 -> 165 -> -228 -> 173 ->
1538 174 -> -168 -> 76 -> -188 -> -281 -> -153 -> 96 ->
1539 -204 65 10.3
1540 -46 183 -> 297 -> -120 -> 222 -> -203 ->
1541 -162 -> 181 -> 298 -> 130 -> 226 -> -171
1542 73 16.2
1543 -14 -205 -> -250 ->
1544 180 -> -284 -> -182
1545 71 34.5
1546 -7 225 ->
1547 -195 -> 209
1548 68
1549 37.3
1550 3 158 ->
1551 -81 -> -161
1552 22
1553 18.8
1554 12 -109 -> 286 ->
1555 -135 -> 300 -> 116
1556 -45 24.3
1557 26 161 ->
1558 80 -> -158
1559 -46
1560 18.8
1561 33 122 ->
1562 -98 -> -154
1563 -69
1564 16.3
1565 35
1566 193
1567 -54 62.4
1568 38 -202 ->
1569 110 -> -202

```

```

1566                                     -52
1567                                26.1
1568    40                                     -116 -> 302 ->
1569        135 -> 280 -> 109
1570                                     61                24.2
1571    43                                     273 -> 122 -> 231 -> -121
1572        -> -251 -> 180 -> 290 -> -182
1573                                     66                19.3
1574    44
1575        -171
1576        -64                63.2
1577    47                                     -166 ->
1578        103 -> -157
1579                                     50
1580                                26.6
1581    48                200 -> -104 -> -137 -> 219 -> 201 ->
1582        -178 -> -198 -> 117 -> 138 -> 256 -> 186
1583                -56                12.8
1584    50                273 -> 122 -> 231 -> 160 -> -205
1585        -> -251 -> 180 -> -284 -> 260 -> -154
1586                55                21.7
1587    53
1588        140
1589        70                62.1
1590    55                -130 -> 304 -> -181 -> -163 ->
1591        203 -> 221 -> 120 -> 301 -> -183
1592                10                20.3
1593    57
1594        199
1595
1596    56                62.0
1597    60                -186 -> -255 -> -138 -> 118 -> 198 ->
1598        177 -> -201 -> -220 -> 137 -> -105 -> -200
1599                42                14.7
1600    62
1601        194
1602        -65                61.5
1603    66                115 -> -291 -> -172 -> -187 -> -229 ->
1604        285 -> 185 -> -233 -> 167 -> 303 -> 134
1605                64                16.3
1606    70                                     206 -> 176
1607        -> -119 -> -215
1608                                     31
1609        31.0
1610
1611    Creating loop unrolling bridges (2025-08-29 02:42:14)
1612    When a SPAdes contig path connects an anchor contig with
1613        the middle contig
1614    of a simple loop, Unicycler concludes that the sequences
1615        are contiguous (i.e.
1616    the loop is not a separate piece of DNA). It then uses the
1617        read depth of the
1618    middle and repeat contigs to guess the number of times to
1619        traverse the loop and
        makes a bridge.

```

1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673

Loop count Start	Loop count Repeat	Loop count Middle	Loop End	Bridge by repeat	by middle
38	-202	110	-52	0.51	0.88
	1	39.4			

Applying bridges (2025-08-29 02:42:14)
Unicycler now applies to the graph in decreasing order of quality. This ensures that when multiple, contradictory bridges exist, the most supported option is used.

Bridge type	Start -> end	Path	Quality
SPAdes	44 -> -64	-171	63.246
SPAdes	-60 -> 62	-199	63.076
SPAdes	35 -> -54	193	62.400
SPAdes	-54 -> 57	131	62.184
SPAdes	53 -> 70	140	62.119
SPAdes	57 -> 56	199	61.983
SPAdes	62 -> -65	194	61.468
SPAdes	-7 -> 68	225, -195, 209	37.263
SPAdes	-14 -> 71	-205, -250, 180, -284, -182	34.518
SPAdes	70 -> 31	206, 176, -119, -215	30.961
SPAdes	47 -> 50	-166, 103, -157	26.582
SPAdes	38 -> -52	-202, 110, -202	26.068
SPAdes	12 -> -45	-109, 286, -135, 300, 116	24.336
SPAdes	40 -> 61	-116, 302, 135, 280, 109	24.162
SPAdes	50 -> 55	273, 122, 231, 160, -205, -251, 180, -284, 260, -154	21.712
SPAdes	55 -> 10	-130, 304, -181, -163, 203, 221, 120, 301, -183	20.315
SPAdes	43 -> 66	273, 122, 231, -121, -251, 180, 290, -182	19.307
SPAdes	3 -> 22	158, -81, -161	18.808
SPAdes	26 -> -46	161, 80, -158	18.802

```

1674 SPAdes          33 -> -69      122, -98, -154
1675                               16.337
1676 SPAdes          66 -> 64       115, -291, -172, -187, -229,
1677       285, 185,          16.315
1678 -233, 167, 303, 134
1679 SPAdes          -46 -> 73      183, 297, -120, 222, -203,
1680       -162, 181,          16.239
1681 298, 130, 226, -171
1682 SPAdes          60 -> 42      -186, -255, -138, 118, 198,
1683       177, -201,          14.735
1684 -220, 137, -105, -200
1685 SPAdes          48 -> -56      200, -104, -137, 219, 201,
1686       -178, -198,          12.791
1687 117, 138, 256, 186
1688 SPAdes          -47 -> 65      -196, -265, 128, -113, 165,
1689       -228, 173,          10.287
1690 174, -168, 76, -188, -281, -153, 96,
1691 -204
1692 Saving ./output/unicycler/004_bridges_applied.gfa
1693
1694 Bridged assembly graph (2025-08-29 02:42:14)
1695 The assembly is now mostly finished and no more structural
1696 changes will be
1697 made. Ideally the assembly graph should now have one
1698 contig per replicon and no
1699 erroneous contigs (i.e. a complete assembly). If there are
1700 more contigs, then
1701 the assembly is not complete.
1702
1703 Saving ./output/unicycler/005_final_clean.gfa
1704
1705 Component      Segments      Links      Length      N50
1706   Longest segment      Status
1707 total          178          240    4,938,037    156,616
1708 617,816
1709 1          155          209    4,670,836    157,869
1710 617,816 incomplete
1711 2          22          31     214,554     60,464
1712 91,038 incomplete
1713 3          1          0       52,647     52,647
1714 52,647 incomplete
1715
1716 Assembly complete (2025-08-29 02:42:14)
1717 Saving ./output/unicycler/assembly.gfa
1718 Saving ./output/unicycler/assembly.fasta
1719 run success
1720 *****
1721 Step 3:
1722 toolname: QUAST
1723 function: Reference-free (or reference-guided) assembly
1724           quality assessment to generate comprehensive assembly
1725           statistics and an interactive HTML report.
1726 description: QUAST (Quality Assessment Tool for Genome
1727               Assemblies) evaluates draft genome assemblies by
1728               computing contiguity, size, and composition metrics and
1729               , when a reference is provided, alignment-based

```


misassembly statistics. Core methods include: (1) reference-free statistics (N50/L50, NG50 if genome size is known, total length, largest contig, number of contigs above thresholds, GC%, ambiguous bases, duplication, and k-mer-based composition summaries), and (2) reference-guided evaluation via fast whole-genome alignments (MUMmer/NUCmer) to report misassemblies, relocations/inversions, indels, and genome fraction. QUAST can optionally map reads back to the assembly using standard short-read aligners (e.g., Bowtie2/BWA, invoked internally) to compute coverage and support-based metrics. Strengths: widely used for bacterial assemblies, produces both machine-readable TSVs and an interactive HTML report with plots; supports multiple assemblies for side-by-side comparison. Limitations: without a suitable reference, misassembly detection is limited to read/coverage-based cues and general contiguity metrics; interpretation of metrics requires context (e.g., expected genome size). For this workflow, QUAST will take the Unicycler contig FASTA and produce the required TSV statistics and an HTML evaluation report, complementing the existing FASTA/GFA outputs.

inputformat: Required: Assembled genome in FASTA (e.g., ./output/unicycler/SRR11874161_unicycler_assembly.fasta). Optional: paired-end FASTQ reads for coverage-based metrics (e.g., ./output/fastp/SRR11874161_fastp_trimmed_R1.fastq and ..._R2.fastq). Optional: reference genome FASTA for alignment-based misassembly analysis.

outputformat: HTML: interactive assembly quality report (plots and summaries); TSV: assembly statistics (e.g., report.tsv with N50, L50, total length, GC%, number of contigs, largest contig), plus additional tab-delimited detail files (e.g., misassemblies.tsv when reference provided). These fulfill the required QC/assembly evaluation reports (HTML) and assembly metrics (TSV).

conference: ['MetaQUAST', 'quast', 'assembly-stats', 'mercury', 'compleasm', 'genomescope', 'jellyfish', 'cami_amber', 'art', 'velvet']

toolname: quast

description: QUAST: assembly quality assessment. Computes contiguity and composition metrics (e.g., total length, N50/L50, largest contig, GC%, number of contigs > thresholds) and generates an interactive HTML report and tabular summaries. Example: given the Unicycler contig FASTA from SRR11874161, QUAST produces reference-free assembly statistics and plots for review.

used_reference_tool: True

toolid: 1

selected_input_files: [{'file_name': './output/unicycler/SRR11874161_unicycler_assembly.fasta', 'file_format': 'FASTA'}]

expected_outputs_info: A results directory containing: HTML report (interactive summary), TSV/TSV tables (e.g., report.tsv with N50, L50, total length, GC%, contig counts), plain-text summaries (report.txt), and figure

```

1782         files (PNG/PDF) for cumulative length and NG/N50 plots.
1783         No reference provided, so outputs are reference-free
1784         metrics only.
1785
1786     installation_command: conda create -n quast -c conda-forge
1787         -c bioconda quast python=3.11 -y && conda activate
1788         quast
1789     setup_command: mkdir -p ./output/quast/
1790     execution_command: bash -lc '
1791     set -euo pipefail
1792     ASM= ./output/unicycler/SRR11874161_unicycler_assembly.
1793         fasta
1794     OUT= ./output/quast
1795     SAMPLE= SRR11874161
1796     [ -s $ASM ]
1797     for f in \
1798         $OUT/${SAMPLE}_quast_report.html \
1799         $OUT/${SAMPLE}_quast_metrics.tsv \
1800         $OUT/${SAMPLE}_quast_summary.txt \
1801         $OUT/${SAMPLE}_quast_report.pdf \
1802         $OUT/${SAMPLE}_quast_metrics_transposed.tsv \
1803         $OUT/${SAMPLE}_quast_Nx_plot.pdf \
1804         $OUT/${SAMPLE}_quast_NGx_plot.pdf \
1805         $OUT/${SAMPLE}_quast_cumulative_plot.pdf \
1806         $OUT/${SAMPLE}_quast_Nx_plot.png \
1807         $OUT/${SAMPLE}_quast_NGx_plot.png \
1808         $OUT/${SAMPLE}_quast_cumulative_plot.png ; do
1809     if [ -e $f ]; then echo Error: output already exists:
1810         $f >&2; exit 1; fi
1811     done
1812     quast.py \
1813     --threads 8 \
1814     --min-contig 200 \
1815     --output-dir $OUT \
1816     $ASM
1817     [ -s $OUT/report.html ] && mv $OUT/report.html $OUT/${
1818     {SAMPLE}_quast_report.html
1819     [ -s $OUT/report.tsv ] && cp $OUT/report.tsv $OUT/${
1820     {SAMPLE}_quast_metrics.tsv
1821     [ -s $OUT/report.txt ] && cp $OUT/report.txt $OUT/${
1822     {SAMPLE}_quast_summary.txt
1823     if [ -s $OUT/report.pdf ]; then cp $OUT/report.pdf
1824     $OUT/${SAMPLE}_quast_report.pdf ; fi
1825     if [ -s $OUT/transposed_report.tsv ]; then cp $OUT/
1826     transposed_report.tsv $OUT/${SAMPLE}
1827     _quast_metrics_transposed.tsv ; fi
1828     for ext in pdf png; do
1829     [ -f $OUT/plots_${ext}/Nx_plot.${ext} ] && cp $OUT/
1830     plots_${ext}/Nx_plot.${ext} $OUT/${SAMPLE}
1831     _quast_Nx_plot.${ext} || true
1832     [ -f $OUT/plots_${ext}/NGx_plot.${ext} ] && cp $OUT/
1833     plots_${ext}/NGx_plot.${ext} $OUT/${SAMPLE}
1834     _quast_NGx_plot.${ext} || true
1835     [ -f $OUT/plots_${ext}/cumulative_plot.${ext} ] && cp
1836     $OUT/plots_${ext}/cumulative_plot.${ext} $OUT/${
1837     SAMPLE}_quast_cumulative_plot.${ext} || true
1838     done
1839     '
1840     2025-08-29 02:57:06

```

```

1836 Creating large visual summaries...
1837 This may take a while: press Ctrl-C to skip this step..
1838 1 of 2: Creating PDF with all tables and plots...
1839 2 of 2: Creating Icarus viewers...
1840 Done
1841
1842 2025-08-29 02:57:06
1843 RESULTS:
1844 Text versions of total report are saved to ./output/quast/
1845 report.txt, report.tsv, and report.tex
1846 Text versions of transposed total report are saved to ./
1847 output/quast/transposed_report.txt, transposed_report.
1848 tsv, and transposed_report.tex
1849 HTML version (interactive tables and plots) is saved to ./
1850 output/quast/report.html
1851 PDF version (tables and plots) is saved to ./output/quast/
1852 report.pdf
1853 Icarus (contig browser) is saved to ./output/quast/icarus.
1854 html
1855 Log is saved to ./output/quast/quast.log
1856
1857 Finished: 2025-08-29 02:57:06
1858 Elapsed time: 0:00:01.509758
1859 NOTICES: 1; WARNINGS: 0; non-fatal ERRORS: 0
1860
1861 Thank you for using QUAST!
1862
1863 run success
1864 *****
1865
1866 Step 4:
1867 toolname: Bowtie2
1868 function: Map the trimmed Illumina paired-end reads back
1869 to the Unicycler assembly to generate high-quality read
1870 -to-contig alignments required for assembly polishing (
1871 e.g., with Pilon) and for downstream coverage/mapping
1872 QC.
1873 description: Bowtie2 is a fast, memory-efficient gapped
1874 short-read aligner based on the Transform and a seed-
1875 and-extend strategy. It is widely used to align
1876 Illumina paired-end reads to a reference, supporting
1877 local or end-to-end alignment modes with quality-aware
1878 scoring and handling of small indels. In bacterial de
1879 novo assembly workflows, Bowtie2 is the standard choice
1880 to map cleaned reads back to assembled contigs,
1881 producing the alignments that polishing tools (e.g.,
1882 Pilon, POLCA) use to detect and correct residual SNP/
1883 indel errors and small misassemblies. It also enables
1884 coverage assessment and mapping statistics for
1885 contamination checks and assembly evaluation. Strengths
1886 : very fast and accurate for short reads, robust paired
1887 -end handling, and good default presets (e.g., --very-
1888 sensitive-local) for polishing. Limitations: not
1889 designed for long reads; highly repetitive regions can
1890 yield multi-mapping reads; large structural variations
1891 are not its focus. Typical usage: build an index from
1892 the Unicycler FASTA (bowtie2-build), align paired-end
1893 trimmed reads (bowtie2 --very-sensitive-local -x index
1894 -1 R1.fastq -2 R2.fastq -S out.sam), then convert/sort/

```

```

1890         index with SAMtools to produce a coordinate-sorted BAM
1891         for input to a polisher like Pilon. This step directly
1892         addresses the current gap in the workflow (no read-to-
1893         assembly alignments yet), enabling the polishing step
1894         that will produce a higher-quality final FASTA.
1895     inputformat: - Reference: FASTA assembly from Unicycler (e
1896         .g., ./output/unicycler/SRR11874161_unicycler_assembly.
1897         fasta)
1898     - Reads: Paired-end trimmed FASTQ from fastp (e.g., ./
1899         output/fastp/SRR11874161_fastp_trimmed_R1.fastq and ./
1900         output/fastp/SRR11874161_fastp_trimmed_R2.fastq)
1901     - Optional: unpaired reads (FASTQ) if present
1902     outputformat: - Primary: SAM file containing read-to-
1903         contig alignments (convertible to BAM/CRAM via SAMtools
1904         )
1905     - Downstream (recommended): coordinate-sorted, indexed BAM
1906         (BAM + BAI) for polishing with Pilon
1907     - Mapping statistics (stderr/log) that can inform coverage
1908         -based QC and contamination checks
1909     Mapping to user's final outputs: while Bowtie2 produces
1910         intermediate alignment files (SAM/BAM) rather than the
1911         final FASTA/GFA/HTML/TSV deliverables, these alignments
1912         are necessary to run a polisher (e.g., Pilon) that
1913         will improve the final FASTA assembly quality and
1914         support comprehensive QC.
1915
1916     conference: ['bowtie2', 'racon', 'bowtie_wrappers', 'pilon
1917         ', 'necat', 'ngmlr', 'minimap2', 'rasusa', 'sickle', '
1918         colibread']
1919
1920     toolname: Bowtie2
1921     description: Function: Map trimmed Illumina paired-end
1922         reads back to the Unicycler assembly to generate high-
1923         quality read-to-contig alignments for polishing and
1924         coverage QC. What it does: Builds an index from the
1925         assembly FASTA and aligns paired reads, producing a SAM
1926         alignment file suitable for conversion to sorted BAM
1927         for tools like Pilon. Example: bowtie2-build
1928         SRR11874161_unicycler_assembly.fasta idx; bowtie2 --
1929         very-sensitive-local -x idx -1
1930         SRR11874161_fastp_trimmed_R1.fastq -2
1931         SRR11874161_fastp_trimmed_R2.fastq -S
1932         SRR11874161_vs_assembly.sam
1933     used_reference_tool: True
1934     toolid: 0
1935     selected_input_files: [{'file_name': './output/unicycler/
1936         SRR11874161_unicycler_assembly.fasta', 'file_format': '
1937         FASTA'}, {'file_name': './output/fastp/
1938         SRR11874161_fastp_trimmed_R1.fastq', 'file_format': '
1939         FASTQ'}, {'file_name': './output/fastp/
1940         SRR11874161_fastp_trimmed_R2.fastq', 'file_format': '
1941         FASTQ'}]
1942     expected_outputs_info: Primary: SAM file of read-to-
1943         assembly alignments. Typically followed by samtools to
1944         produce a coordinate-sorted BAM (BAM + BAI) for
1945         polishing (e.g., Pilon) and coverage/mapping QC.
1946
1947     Returning block of 716588 for bucket 7
1948     Exited Ebwt loop

```

```

1944     fchr[A]: 0
1945     fchr[C]: 1218875
1946     fchr[G]: 2468300
1947     fchr[T]: 3712787
1948     fchr[$]: 4936436
1949     Exiting Ebwt::buildToDisk()
1950     Returning from initFromVector
1951     Wrote 5845162 bytes to primary EBWT file: ./output/Bowtie2
1952           /SRR11874161_Bowtie2_index.rev.1.bt2.tmp
1953     Wrote 1234116 bytes to secondary EBWT file: ./output/
1954           Bowtie2/SRR11874161_Bowtie2_index.rev.2.bt2.tmp
1955     Re-opening _in1 and _in2 as input streams
1956     Returning from Ebwt constructor
1957     Headers:
1958     len: 4936436
1959     bwtLen: 4936437
1960     sz: 1234109
1961     bwtSz: 1234110
1962     lineRate: 6
1963     offRate: 4
1964     offMask: 0xffffffff0
1965     ftabChars: 10
1966     eftabLen: 20
1967     eftabSz: 80
1968     ftabLen: 1048577
1969     ftabSz: 4194308
1970     offsLen: 308528
1971     offsSz: 1234112
1972     lineSz: 64
1973     sideSz: 64
1974     sideBwtSz: 48
1975     sideBwtLen: 192
1976     numSides: 25711
1977     numLines: 25711
1978     ebwtTotLen: 1645504
1979     ebwtTotSz: 1645504
1980     color: 0
1981     reverse: 1
1982     Total time for backward call to driver() for mirror index:
1983           00:00:02
1984
1985     run success
1986     *****
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997

```