BioMANIA: Simplifying bioinformatics data analysis through conversation

Zhengyuan Dong[1] Victor Zhong[1]
Yang Young Lu[1*]
[1]Cheriton School of of Computer Science, University of Waterloo

Abstract

The burgeoning field of bioinformatics has been revolutionized by the rapid growth of omics data, providing insights into various biological processes. However, the complexity of bioinformatics tools and the rapidly evolving nature of data analysis pipelines present significant challenges for researchers, especially those lacking extensive programming expertise. To address these challenges, we introduce BioMANIA, an artificial intelligence-driven, natural language-oriented bioinformatics data analysis pipeline. BioMANIA comprises two key components: a ChatBot generation pipeline and a user-friendly ChatBot back-end service. The generation pipeline takes as input an open-source (*e.g.,* hosted in GitHub) and well-documented (*e.g.,* hosted in ReadTheDocs) Python tool, extracting API attributes and generating synthetic instructions that train a ChatBot to understand and perform specific data analysis tasks. We identified 11 common issues to provide a practical guideline for designing more ChatBot-compatible tools, which we categorized into five groups, while analyzing 12 well-documented open-source Python tools across various bioinformatics settings. The ChatBot service then assists users in selecting the appropriate analysis API and parameters, significantly reducing the programming barrier. We applied BioMANIA to analyze single-cell gene expression data, demonstrating its effectiveness in simplifying complex omics data analysis. BioMANIA has the potential to transform the landscape of bioinformatics research, making data analysis more accessible and accelerating discoveries in the field. [1]

1 Introduction

Advancements in high-throughput sequencing technologies have enabled researchers to gather extensive omics data, including genomics, transcriptomics, epigenomics, proteomics, metabolomics, and microbiomics, from various tissues, organs, individuals, and species. For example, consortium projects such as The Cancer Genome Atlas (TCGA) [1] and Encyclopedia of DNA Elements (ENCODE) [2] have provided open access to genomic, transcriptomic and epigenomic information from a diverse group of samples. While the abundance of omics data has fueled interest in comprehending molecular mechanisms and therapeutic applications, it also presents significant challenges for researchers in deriving meaningful insights from data analysis.

Over the years, substantial efforts have been made to address these data analysis challenges. These endeavors yield meaningful insights from two perspectives [3]: (1) computational researchers

*Corresponding Author. Email: yanglu@uwaterloo.ca
[1]The GNU licensed source code of BioMANIA will be available at https://github.com/batmen-lab/BioMANIA.

create novel methods and tools to analyze biological data, and (2) experimental researchers employ existing tools and pipelines for analyzing and understanding the data. At first, bioinformatics tools were written as interior scripts in programming languages like Perl, Ruby, R, Python, and Java. Subsequently, specialized bioinformatics programming frameworks emerged to advance omics data analysis and visualization, including BioPerl [4], BioRuby [5], Bioconductor [6], Biopython [7], and BioJava [8]. However, using these tools demands significant bioinformatics programming skills, which some experimental researchers may not readily possess.

To reduce the programming barrier, several tools have been created to enable users to explore and visualize omics data without extensive programming skills. Briefly speaking, each tool provides an interactive portal for omics data exploration. For example, Galaxy [9] is a module hub with pre-assembled bioinformatics pipelines for interactive large-scale genome analysis. Furthermore, the cBio Cancer Genomics Portal [10] allows interactive exploration and visualization of preloaded multidimensional cancer genomics datasets. However, these tools are constrained to predefined analyses, often requiring users to learn complex interfaces and procedures. Worse still, existing pipelines rapidly evolve with algorithm advancements, leading to increased complexity. For example, the single-cell RNA-seq (scRNA-seq) data analysis pipeline reveals its complexity through a series of steps, including quality control, feature selection, dimensionality reduction, cell clustering, and cell annotation [11]. Additionally, the pipeline extends to various downstream analyses such as batch correction, trajectory inference, and cell-cell communication [12]. In summary, effectively utilizing diverse omics data demands significant professional skills and domain-specific expertise, which can be a barrier for researchers with limited experience in data analysis.

From a practitioner's perspective, ideal bioinformatics data analysis should include the following properties: (1) Standardized and automated pipelines for shareable and reproducible analysis [13], and (2) User-friendly, codeless analysis to overcome the learning barrier for researchers with limited programming skills [14]. Coincidentally, large language models (LLMs) like GPT-3 [15] and GPT-4 [16] have made significant advancements in various natural language processing tasks. LLMs have shown superior capabilities in human-like conversation [17], mathematical reasoning [18], code generation [19], and more. In the field of bioinformatics, a domain beyond natural language processing, they prove valuable in bioinformatics education [20, 21] and programming tasks [22, 3]. We believe that achieving desirable bioinformatics data analysis with shareability, reproducibility, user-friendliness, and codelessness is effectively attainable through natural language

In this paper, we surmount the bioinformatics data analysis barrier using natural languages and propose an artificial intelligence (AI)-driven, natural language-oriented bioinformatics data analysis pipeline, named BioMANIA (BioinforMatics dAta aNalysIs through conversAtion), as illustrated in Figure 1. At a high level, BioMANIA comprises two core subsystems: a ChatBot generation pipeline and a user-friendly ChatBot service. The ChatBot generation pipeline takes as input an open-source (*e.g.,* hosted in GitHub) and well-documented (*e.g.,* hosted in ReadTheDocs) Python tool. The tool's documentation typically comprises a carefully curated list of open-access APIs and tutorials that demonstrate data analysis using the tool. BioMANIA first employs an Abstract Syntax Tree (AST) parser to extract API attributes, including function description, input parameters, and return values. BioMANIA learns from tutorials, identifies the interplay between API usage, and aggregates APIs into meaningful functional ensembles. BioMANIA then prompts LLMs to comprehend the API functionalities and generates synthetic instructions corresponding to API calls. These instruction-API pairs train an API retriever to narrow down candidate APIs to the top-k for a given instruction and an API predictor to predict the exact API from the retrieved

subset. The ChatBot service takes the user's data and instructions as input and determines whether the user is asking an API-specific data analysis question or engaging in chit-chat. BioMANIA next employs the API retriever to filter candidate APIs and the API predictor to select the target API. BioMANIA then instructs LLMs to predict the parameters for the target API call and verify their correctness and completeness. Finally, BioMANIA processes the target API call using a built-in Python interpreter and sends the results to the user along with a detailed report. We applied BioMANIA to codelessly analyze single-cell gene expression data using Scanpy [23], showcasing its empirical utility. BioMANIA has the potential to usher in a new research paradigm, simplifying the exploration of omics data and accelerating bioinformatics research.

2 Methods

The BioMANIA ChatBot generation takes as input a new open-source (*e.g.,* hosted on GitHub) and well-documented (*e.g.,* hosted on ReadTheDocs) Python tool and creates an automated ChatBot that translates natural language user requests to executable API calls.

2.1 API extraction

First, BioMANIA employs an Abstract Syntax Tree (AST) parser to analyze the source code and extract potential APIs. For each API, we further extract information about its requirements (parameter list) and operation (name, description, return value) to utilize the API effectively. Parameters can be either required or optional, each with its parameter name, description, data type, and default value. Second, to filter out low-quality and unreliable APIs from direct source-code analyses, we further remove APIs without documentation and APIs that are not executable.

When there are tutorials available that guide users how to combine APIs to accomplish complex tasks, we additionally extract compositional usage of APIs from these tutorials. Specifically, we form composite APIs that correspond to code blocks in the tutorial [24]. Similar to standard APIs, we infer the parameters and return value of the composite API from the tutorial text. In order to determine the description of the composite API, we prompt LLMs (*e.g.,* GPT-4) to summarize the composite API's functionality concisely given descriptions of APIs used in the tutorial code block. Table A.1 and Table A.2 show examples of detailed prompts and composite APIs.

2.2 Instruction generation

Learning to predict API calls from user instructions typically requires a large-scale dataset of instruction-API pairs [19]. Instead of collecting such data through human annotation, which is expensive and time-consuming, we automatically synthesize language instructions given information about an API [25]. Specifically, we provide LLMs with the API name, description, and parameter list as input and prompt them to generate five varied, detailed, and verbose instructions for making API calls. To improve robustness generalize to different usage scenarios, we additionally generate five precise and concise instructions to diversify the instructions. In total, we synthesize ten language instructions for each API. As shown in Figure 1 (A), this dataset is used to train a model that predicts which API to use given user instruction. Tables A.3 and A.4 show detailed prompts and examples of instruction generation. These instruction examples are also re-purposed to distinguish between chit-chat and API-specific data analysis questions from the user (Section 2.3).

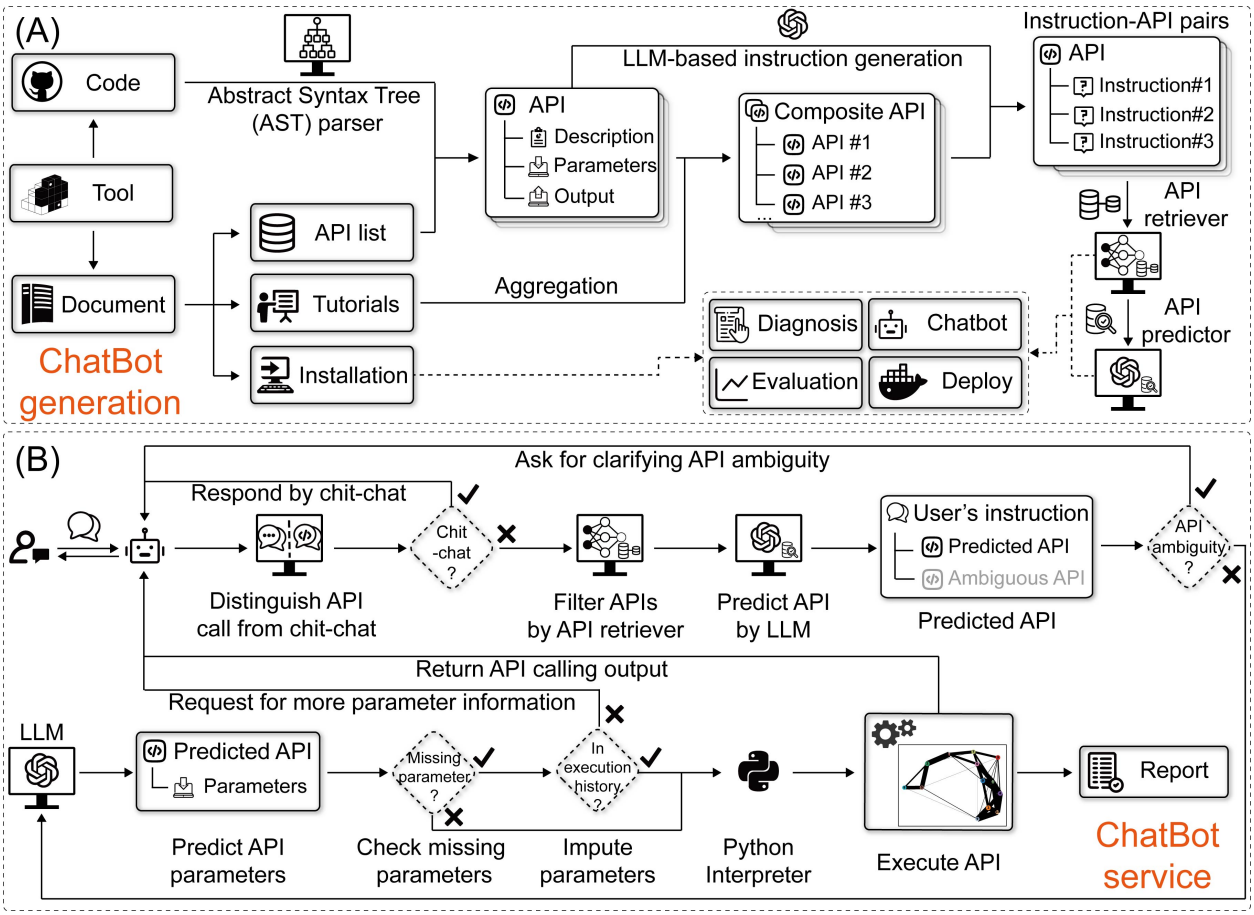

Figure 1: **Overview of BioMANIA.** BioMANIA comprises a ChatBot generation pipeline and a user-friendly ChatBot service. (A) The BioMANIA ChatBot generation pipeline first learns API information and their interplay from the source code and documentation. BioMANIA then prompts LLMs to comprehend the API functionalities and generates synthetic instructions corresponding to API calls. These instruction-API pairs train an API retriever to narrow down candidate APIs to the top-k for a given instruction and an API predictor to predict the exact API from the retrieved subset. Finally, BioMANIA creates and deploys a tool-specific ChatBot, providing a diagnosis report with documentation improvement suggestions and an evaluation report on the quantitative performance of each step. (B) The BioMANIA ChatBot service takes the user's data and instructions as input and determines whether the user is asking an API-specific data analysis question or engaging in chit-chat. BioMANIA next employs the API retriever to filter candidate APIs and the API predictor to select the target API. It is worth mentioning that BioMANIA identifies potentially ambiguous APIs during ChatBot generation and requests user clarification for ambiguity. BioMANIA then instructs LLMs to predict the required parameters for the target API call and verify their correctness and completeness. If the missing parameters can not be filled by the context generated from API execution history, BioMANIA will request more detailed parameter information from the user. Finally, BioMANIA processes the target API call using a built-in Python interpreter and sends the results to the user along with a detailed report.

2.3 Chit-chat recognition

In natural conversation, a non-trivial portion of dialogue do not express user intent — they correspond to greetings or remarks that do not request for information or program execution [26]. Consequently, we incorporate a chit-chat detection stage to identify whether an user utterance is inquiring about API-encoded data analysis or engaging in casual chit-chat. To distinguish between API calls and chit-chat, we compile a substantial corpus of open-domain conversational samples from two existing chit-chat datasets: Topical-Chat [27] and Qnamaker [28]. Both chit-chat samples and API-encoded instructions are represented as TF-IDF embeddings [29], which involve vectorizing the TF-IDF (Term Frequency-Inverse Document Frequency) scores, a widely-used metric for word importance, for every word in the dictionary. We calculate centroids for three groups (*i.e.,* instructions, Topical-Chat samples, and Qnamaker samples) and use them to build a nearest centroid classifier for chit-chat classification. Namely, for a new user instruction, we calculate the cosine similarity between its TF-IDF embedding and these centroids, and predict the label based on the closest matched centroid. Despite its simplicity, we found that the nearest centroid classifier performs accurately and efficiently in practice (See Section 3.3.1 for more details). Finally, the chit-chat classification results in two different responses: initiating an API call or providing a chit-chat response. In a standard chit-chat scenario, the ChatBot employs a LLM to engage with users. Otherwise, the ChatBot will identify and select the most suitable API to use (Section 2.4).

2.4 API prediction

Selecting the most suitable API from the user instruction is challenging due to the potential abundance of API candidates. To address this issue, BioMANIA first prunes candidate APIs with a embedding-based retriever, and then selects the target API with long-context predictor. We use a smaller embedding-based model for the retriever as it allows us to scale to large numbers of API candidates. The API retriever encodes the instruction and API document (including its name, description, and parameter list) as embeddings, in which the instruction-API relevance is determined by the cosine similarity between two embeddings. To train the retriever, we fine-tune a BERT-base model [30] to maximize the similarity of matching API-instruction pairs and minimize the similarity of randomly sampled negative API-instruction pairs.

After retrieval, the API predictor searches the retrieved API shortlist to find the best match. For this purpose, we frame API prediction as an in-context API selection task. Specifically, we provide the API shortlist along with multiple demonstrations, each consisting of a synthetic instruction and its corresponding API information. These demonstrations help LLMs (*e.g.,* GPT-4) understand the intrinsic connections between instructions and APIs, facilitating accurate API prediction. We select demonstration samples based on instruction similarity from the training set, chosen by the API retriever [31]. In cases where there exists ambiguous APIs with similar descriptions or identical names, we prompt users for clarification to determine the target API. It is worth mentioning that during API extraction, we encountered issues related to API design, such as different APIs sharing similar descriptions or identical names, leading to ambiguity and potentially impacting API prediction accuracy (Refer to Figure 4 and Section 3.3.2 for details). To address ambiguity, we create an ambiguity list for each API, which prompts user clarification to determine the target API. Tables A.5 and A.6 show detailed prompts and examples of API prediction.

2.5 API execution

To produce an executable API call, we additionally predict the values of required and optional parameters given the predicted API call. For each parameter, we provide LLMs with the user instruction, the predicted API (name and description), the parameter under consideration (name, description, data type, and default value), and a list of potential values. The LLMs will either select the parameter's value from the user instruction or return the default value if no prediction is made. We use a placeholder for the required parameter as a pseudo default value to prompt the user for additional parameter-related information. We create the list of potential values automatically by mining variables from the stack traces of executed API calls whose type matches the parameter type. Tables A.7 and A.8 show detailed prompts and examples of API parameter prediction.

After an API is executed by a Python interpreter, BioMANIA generates final responses from three perspectives. First, BioMANIA integrates all information from previous stages into a concise summary, encompassing API prediction and execution from API execution. Second, BioMANIA presents the execution results in a user-friendly format, like a plotting figure or a snapshot of the data matrix. Third, BioMANIA organizes the API calls into a downloadable script for reproducibility and generates a comprehensive report recording the entire data analysis process. Section 3.2 shows detailed case studies of using BioMANIA for natural language-based data analysis.

3 Results

3.1 Experimental setup

To evaluate the effectiveness of BioMANIA, we employed a widely-used tool, Scanpy [23], for analyzing single-cell gene expression data across a range of functionalities. Scanpy offers 166 Python-based APIs for a range of analyses, including preprocessing, visualization, clustering, trajectory inference, and differential expression testing. Based on Scanpy tutorials, we defines an additional 16 API ensembles that combines multiple APIs to accomplish complex tasks. We used GPT-4 to generate 5 detailed and 5 concise synthetic instructions with various tones for each of the 182 APIs, following the prompts shown in Table A.3. For each API, 8 out of the 10 synthetic instructions are assigned to the training split for training the API retriever and API predictor, while the remaining 2 are used for evaluation purposes. Refer to Figure 3(D) for illustrative examples of synthetic instructions generated by LLMs for a randomly chosen API. Furthermore, we involved human annotators to manually label one instruction for each API, creating an independent test set.

3.2 Scanpy ChatBot reliably conducts single-cell data analysis

We evaluated the effectiveness of Scanpy ChatBot through several demonstrations. For example, as shown in Figure 2(A), a user can upload a 10x-Genomics-formatted dataset in the input area and instruct the ChatBot to parse it. If the instruction lacks sufficient detail, especially when multiple APIs are available for different 10x-Genomics formats, the ChatBot will list potential API candidates and request user clarification to identify the target API. Once the target API and required parameters are identified, the ChatBot will execute the API using a Python interpreter and provide the user with three types of information in response: (1) A concise summary of the API call. (2) The execution results in a user-friendly format, such as a snapshot of the data matrix or a plotting figure. (3) Detailed API calls for downloadable and reproducible purposes. For another example, as shown in Figure 2(B), a user can instruct the Chatbot to perform a complex task

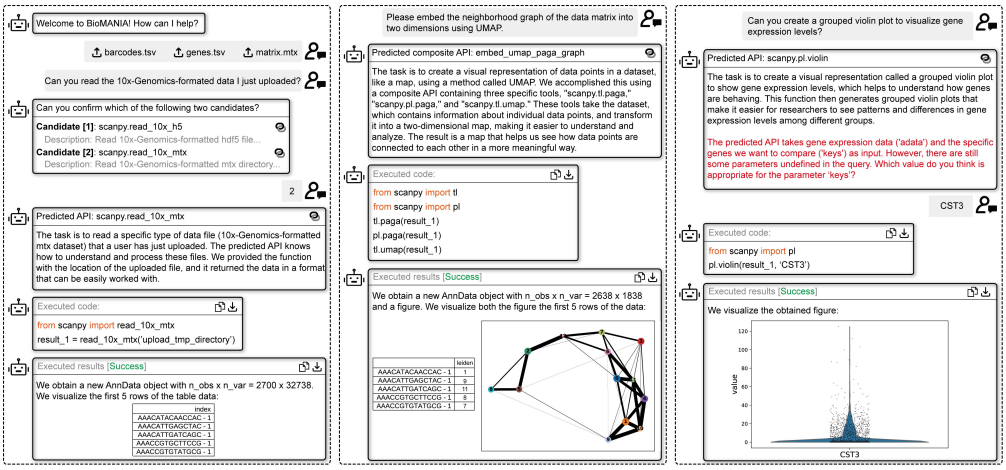

Figure 2: **Illustrative examples of using the Scanpy ChatBot for single-cell data analysis.** These examples are thoughtfully selected to encompass representative scenarios, including API ambiguity clarification, requests for additional parameter information, and predicting API ensembles. (A) The ChatBot assists users in parsing uploaded datasets with a specific format. (B) The ChatBot assists users in projecting and visualizing the data points from the uploaded datasets. (C) The ChatBot assists users in qualitatively understanding the expression level of a gene of interest.

that requires the interaction of multiple APIs. The Chatbot will automatically map the user's instruction to a composite API identified from tool tutorials that illustrate how multiple APIs can be used together to achieve meaningful functionalities. In this scenario, the Chatbot identifies an complex task consisting of three subtasks: generating the neighborhood graph from the uploaded data, plotting the generated neighborhood graph, and embedding the neighborhood graph into a two-dimensional space. In the final example, as shown in Figure 2(C), a user can request the ChatBot to qualitatively display the expression level of a gene of interest but neglect to mention the gene's name in the conversation. The Chatbot accurately predicts the target API for gene expression display but encounters an issue due to the absence of the gene name. As a result, the Chatbot explains to the user what is required to proceed and requests additional parameter information from the user. Figure A.1 shows more examples of using the Scanpy ChatBot.

3.3 Analyses

3.3.1 BioMANIA accurately handles each step in Scanpy ChatBot service

We assessed the effectiveness of BioMANIA in each stage of the Scanpy ChatBot, starting from the user initiating the conversation to the end of the conversation. We first investigated BioMANIA's performance in distinguishing between user queries related to API-encoded data analysis and casual chit-chat. We randomly sampled an equal number of chit-chat samples from two existing chit-chat datasets, Topical-Chat [27] and Qnamaker [28]. As shown in Figure 3(A), the TF-IDF embeddings of chit-chat samples and API-encoded instructions distinctly separate in the t-SNE projection [32]. The notable separation allows a simple nearest centroid classifier to achieve a 98.2% classification accuracy on the test data. After initiating an API call, we examined BioMANIA's performance in identifying the top-3 candidate APIs by utilizing a API Retriever fine-tuned from the BERT-BASE

model [30]. We assessed the retrieval performance by comparing it to two baseline methods: the untuned BERT-BASE model and BM25 [33], a commonly used deterministic relevance measure for document comparison. As shown in Figure 3(B), the fine-tuned API Retriever attains an accuracy of 99.5% for synthetic instructions and 97.8% for human-annotated instructions in shortlisting the target API within the top 3 candidates. The slightly lower performance in human-annotated instructions is anticipated, as the test synthetic instructions are generated by GPT-4 together with the training instructions. In contrast, the BM25-based API Retriever, which lacks access to training data, achieves an accuracy of 70.9% for synthetic instructions and 63.7% for human-annotated instructions, respectively. Lastly, we examined BioMANIA's performance in predicting the target API from the shortlist. We assessed the prediction performance from multiple settings, considering factors such as whether to use the LLM for prediction or not, whether to use the API Retriever or not, whether to formulate the problem as a prediction task or a sequence generation task (by generating the API from scratch), and whether to use GPT-3.5 or GPT-4. As shown in Figure 3(C), the in-context API classification model based on GPT-4 (Section 2.4) achieves an accuracy of 86.7% for synthetic instructions and 86.8% for human-annotated instructions in predicting the target API from the top 3 candidates. It's worth noting that a noticeable portion of misclassifications is caused by the presence of ambiguous APIs by design, which BioMANIA identifies during ChatBot generation. After ruling out the ambiguous APIs, the accuracy increases to 91.3% for both synthetic instructions and human-annotated instructions. Refer to Figure 3(E) for illustrative examples of API retrieval and prediction given a user instruction. Furthermore, we observed that: (1) Utilizing the LLM for prediction significantly outperforms multi-label classification for prediction. (2) Framing the problem as a prediction task yields better results than when formulated as a sequence generation task. (3) Employing the API Retriever helps filter out incorrect answers, thereby increasing prediction accuracy. (4) GPT-4 better comprehends in-context demonstrations and predicts the target API more effectively than GPT-3.5.

3.3.2 Common issues encountered by BioMANIA

To evaluate the robustness of BioMANIA, we collected 12 well-documented open-source Python tools across five different settings: general bioinformatics analysis, genomics, transcriptomics, microbiomics, and proteomics (Table A.9). While analyzing these tools, we identified 11 common issues throughout the BioMANIA ChatBot generation pipeline. We categorized these common issues into five groups, each based on their functionality in BioMANIA ChatBot generation pipeline, as shown in Figure 4. We also generate a diagnosis report based on these issues, providing a practical guideline for designing more ChatBot-compatible tools.

We next break down the issue categories and discuss each issue in the order it occurs in the ChatBot generation pipeline. We first observed that some tools (*e.g.,* ETE toolkit [34]) could not be smoothly installed following the documentary instructions, a fact confirmed by their Github issues. After installation, some tools (*e.g.,* Scanpy [23]) rely on additional dependencies not specified in the requirements for running the analysis. BioMANIA uses the curated list of APIs in the tool's documentation to guide API extraction, but some tools (*e.g.,* QIIME 2 [35]) lack this curated list in their documentation. After API extraction, we performed basic functionality tests as a double-check. However, some tools (*e.g.,* Pyteomics [36])provided APIs that were inconsistent between the documentation and source code, leading to operational failures. Worse still, we identified improper API design that could hinder API call prediction from user instructions. Some tools (*e.g.,* Scanpy [23]) have different APIs with similar or identical descriptions or names. Also, some

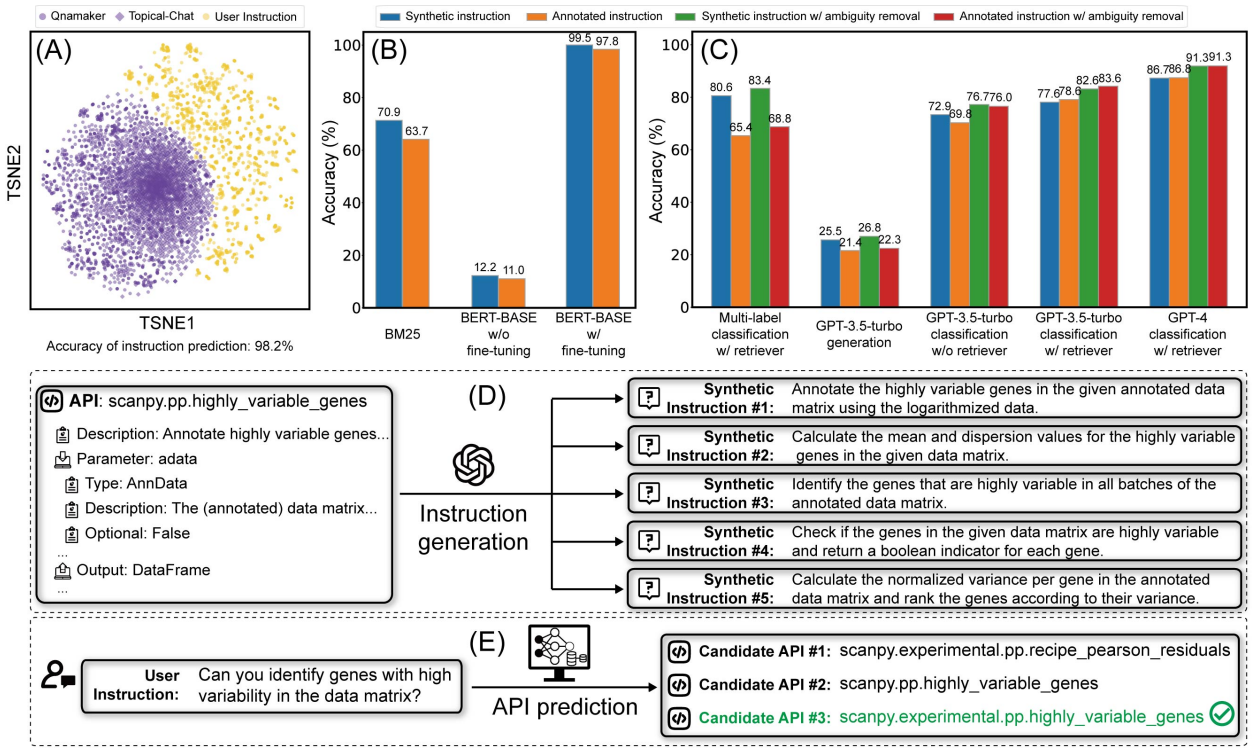

Figure 3: **Evaluating the effectiveness of BioMANIA in each stage of the Scanpy ChatBot.** (A) The t-SNE projection reveals distinct separation between chit-chat and API-encoded instructions based on TF-IDF embeddings. (B) The performance of the fine-tuned API Retriever in identifying the top-3 candidate APIs in comparison with two baselines (BM25 and BERT-BASE without fine-tuning). (C) The performance of API prediction in different settings, considering factors such as the use of the LLM for prediction, the API Retriever, problem formulation (prediction or sequence generation), and the choice between GPT-3.5 and GPT-4. (D) Illustrative examples of synthetic instructions generated by LLMs for a randomly chosen API. (E) Illustrative examples of API retrieval and prediction given a user instruction chosen to match the API in (D).

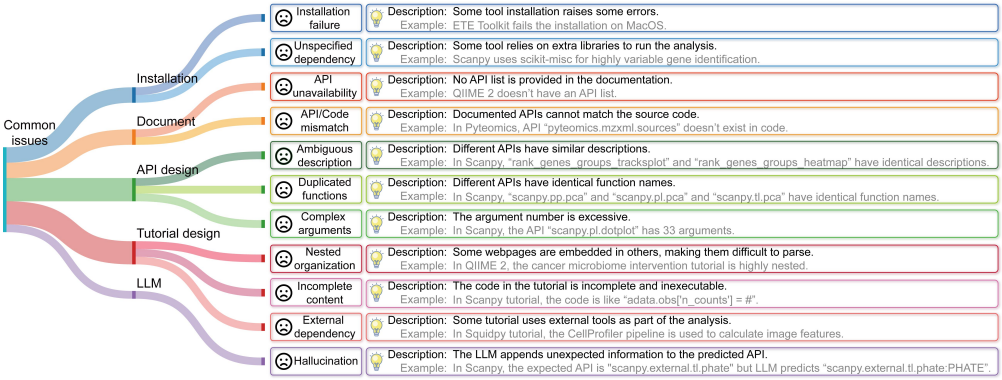

Figure 4: **Common issues encountered by BioMANIA in analyzing 12 Python tools.** These issues are categorized into five groups, each based on their functionality in BioMANIA ChatBot generation pipeline. We offer a concise description for each issue, along with a real example we have encountered in existing tools.

tools (*e.g.,* Scanpy [23]) have very complex API designs with up to dozens of parameters, which can overwhelm the parameter prediction process. BioMANIA examines the use of multiple APIs in tutorials to generate API ensembles, a process that critically depends on the quality of the tutorials. However, improperly designed tutorials in some tools complicate API ensemble generation, including complex and nested webpage design (*e.g.,* QIIME 2 [35]), incomplete and inexecutable code (*e.g.,* Scanpy [23]), and external tool dependencies (*e.g.,* Squidpy [37]). Lastly, we observed hallucination in LLM-based API prediction, where LLMs generate answers that do not match the provided information. For example, we provided a shortlist of candidate APIs for LLM to choose from, but LLM ended up creating an API not in the shortlist.

4 Discussion and conclusion

In this work, we introduced BioMANIA, an AI-driven, natural language-oriented bioinformatics data analysis pipeline aimed at reducing the programming barrier for experimental researchers. The key novelty of BioMANIA includes two aspects: (1) BioMANIA is designed to be versatile, capable of seamlessly converting any well-documented, off-the-shelf open-source Python tool into a Chatbot. BioMANIA establishes a standardized pipeline for extracting API attributes and generating synthetic instructions to train a ChatBot in understanding and performing specific data analysis tasks. (2) BioMANIA summarizes a collection of Python tools and identifies 11 common issues within 5 categories, offering a practical guideline for the future design of more ChatBot-compatible tools. When generating and deploying a tool-specific ChatBot, BioMANIA provides a diagnosis report with documentation improvement suggestions and an evaluation report concerning the quantitative performance of each step. We applied BioMANIA to analyze single-cell gene expression data, demonstrating its effectiveness in simplifying complex omics data analysis. As high-throughput omics technologies continue to advance, the vast amount of omics data has brought forth a golden era for biomedical research. However, it has also presented us with huge challenges in analyzing this data. BioMANIA marks an initial effort to streamline bioinformatics tasks through conversation. We anticipate that BioMANIA will reshape the landscape of bioinformatics research, enhancing data analysis accessibility and expediting discoveries in the field.

References

[1] Weinstein, J. N. et al. The cancer genome atlas pan-cancer analysis project. Nature Genetics **45**, 1113–1120 (2013).

[2] ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. Nature **489**, 57–74 (2012).

[3] Zhou, J. et al. Automated bioinformatics analysis via AutoBA. bioRxiv 2023–09 (2023).

[4] Stajich, J. E. et al. The Bioperl toolkit: Perl modules for the life sciences. Genome Research **12**, 1611–1618 (2002).

[5] Goto, N. et al. BioRuby: bioinformatics software for the Ruby programming language. Bioinformatics **26**, 2617–2619 (2010).

[6] Gentleman, R. C. et al. Bioconductor: open software development for computational biology and bioinformatics. Genome Biology **5**, 1–16 (2004).

[7] Cock, P. J. et al. Biopython: freely available python tools for computational molecular biology and bioinformatics. Bioinformatics **25**, 1422 (2009).

[8] Holland, R. C. et al. BioJava: an open-source framework for bioinformatics. Bioinformatics **24**, 2096–2097 (2008).

[9] Giardine, B. et al. Galaxy: a platform for interactive large-scale genome analysis. Genome Research **15**, 1451–1455 (2005).

[10] Cerami, E. et al. The cBio cancer genomics portal: an open platform for exploring multidimensional cancer genomics data. Cancer Discovery **2**, 401–404 (2012).

[11] He, J., Lin, L. & Chen, J. Practical bioinformatics pipelines for single-cell RNA-seq data analysis. Biophysics Reports **8**, 158 (2022).

[12] Nayak, R. & Hasija, Y. A hitchhiker's guide to single-cell transcriptomics and data analysis pipelines. Genomics **113**, 606–619 (2021).

[13] Wratten, L., Wilm, A. & Göke, J. Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. Nature Methods **18**, 1161–1168 (2021).

[14] Li, J., Chen, H., Wang, Y., Chen, M.-J. & Liang, H. Next-generation analytics for omics data. Cancer Cell **39**, 3–6 (2021).

[15] Brown, T. B. et al. Language models are few-shot learners. Advances in Neural Information Processing Systems **33**, 1877–1901 (2020).

[16] Bubeck, S. et al. Sparks of artificial general intelligence: Early experiments with GPT-4. arXiv preprint arXiv:2303.12712 (2023).

[17] Touvron, H. et al. LLaMA: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023).

[18] Wei, J. et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems **35**, 24824–24837 (2022).

[19] Chen, M. et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021).

[20] Shue, E. et al. Empowering beginners in bioinformatics with ChatGPT. bioRxiv 2023–03 (2023).

[21] Tu, X., Zou, J., Su, W. J. & Zhang, L. What Should Data Science Education Do with Large Language Models? arXiv preprint arXiv:2307.02792 (2023).

[22] Piccolo, S. R., Denny, P., Luxton-Reilly, A., Payne, S. & Ridge, P. G. Many bioinformatics programming tasks can be automated with ChatGPT. arXiv preprint arXiv:2303.13528 (2023).

[23] Wolf, F. A., Angerer, P. & Theis, F. J. SCANPY: large-scale single-cell gene expression data analysis. Genome Biology **19**, 1–5 (2018).

[24] Iyer, S., Cheung, A. & Zettlemoyer, L. Learning programmatic idioms for scalable semantic parsing. arXiv preprint arXiv:1904.09086 (2019).

[25] Wang, Y. et al. Self-Instruct: Aligning language model with self generated instructions (2022).

[26] Stewart, R., Danyluk, A. & Liu, Y. Off-topic detection in conversational telephone speech. In Proceedings of the Analyzing Conversations in Text and Speech, 8–14 (2006).

[27] Gopalakrishnan, K. et al. Topical-Chat: Towards Knowledge-Grounded Open-Domain Conversations. In Proceedings of Interspeech, 1891–1895 (2019). URL `http://dx.doi.org/10.21437/Interspeech.2019-3079`.

[28] Agrawal, P. et al. Qnamaker: Data to bot in 2 minutes. In Companion Proceedings of the Web Conference, 131–134 (2020).

[29] Ramos, J. Using TF-IDF to determine word relevance in document queries. In Proceedings of the First Instructional Conference on Machine Learning, vol. 242, 29–48 (Citeseer, 2003).

[30] Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In Annual Conference of the North American Chapter of the Association for Computational Linguistics, 4171–4186 (2019).

[31] Robinson, J., Chuang, C.-Y., Sra, S. & Jegelka, S. Contrastive learning with hard negative samples. arXiv preprint arXiv:2010.04592 (2020).

[32] der Maaten, L. V. & Hinton, G. Visualizing data using t-SNE. Journal of Machine Learning Research **9**, 85 (2008).

[33] Robertson, S., Zaragoza, H. et al. The probabilistic relevance framework: BM25 and beyond. Foundations and Trends® in Information Retrieval **3**, 333–389 (2009).

[34] Huerta-Cepas, J., Serra, F. & Bork, P. ETE 3: Reconstruction, analysis, and visualization of phylogenomic data. Molecular Biology and Evolution **33**, 1635–1638 (2016).

[35] Bolyen, E. et al. Reproducible, interactive, scalable and extensible microbiome data science using QIIME 2. Nature Biotechnology **37**, 852–857 (2019).

[36] Goloborodko, A. A., Levitsky, L. I., Ivanov, M. V. & Gorshkov, M. V. Pyteomics–a python framework for exploratory data analysis and rapid software prototyping in proteomics. Journal of The American Society for Mass Spectrometry **24**, 301–304 (2013).

[37] Palla, G. et al. Squidpy: a scalable framework for spatial omics analysis. Nature Methods **19**, 171–178 (2022).

A Appendix

Table A.1: Prompt for API ensemble in BioMANIA

Task Description:
You are an expert in Python programming. Your task is to write the docstring for the given information of an invisible function. Interpret the assigned inputs and return variables in the docstring.

The description of used APIs inside this code is: {API_description}
The input and output parameter information is as below:
- Parameters: {func_inputs}
- Returns: {func_outputs}
- The other description associated with the code is: {description_text}
- Please extract the core information in 1-2 sentences and polish it. Docstring description should only use 1-2 sentences.

Your Response format is detailed docstring. Please do not include other information except for response information, in reStructuredText format. Never include specific API information in description.

Table A.2: Input and output examples for API ensemble in BioMANIA

# **Example Input**:
API_description="""
API1: Normalize each cell by total counts over genes, so that every cell has the same total count after normalization.
API2: Logarithmize the data matrix.
API3: Principal component analysis [Pedregosa11].
API4: Compute a neighborhood graph of observations [Wolf19].
API5: Embed the neighborhood graph using UMAP [McInnes18].
API6: Cluster cells into subgroups [Levine15], [Traag17].
"""
func_inputs="adata ('AnnData'): An annotated data matrix. "
func_outputs="'AnnData': An annotated data matrix. "
description_text="""
We will perform standard preprocessing and dimensionality reduction. The experiment consisted in measuring gene expression counts from a single cell type (cultured U2-OS cells). Clusters consist of cell states at different stages of the cell cycle.
"""

Example Output:
"""
Preprocesses and reduces dimensionality of the input annotated data matrix.

This function takes an annotated data matrix as input and performs standard preprocessing steps, including normalizing each cell by total counts over genes, logarithmizing the data matrix, and applying Principal Component Analysis (PCA). It then computes a neighborhood graph of observations using a specified API and embeds the graph using UMAP. Finally, it clusters cells into subgroups based on their gene expression profiles.

Parameters:
adata ('AnnData'): An annotated data matrix representing gene expression counts from single cells.

Returns:
'AnnData': An annotated data matrix after preprocessing and dimensionality reduction.

Note:
The experiment involved measuring gene expression counts from cultured U2-OS cells. The clusters obtained correspond to different stages of the cell cycle.

"""

Table A.3: Prompt for instruction generation in BioMANIA

Input:
Given API information starts:
{Input}
Given API information ends

Task Description:
You are provided with one API function, its description, its parameters, and its return information for the API function. Your task involves creating 5 varied, innovative, and detailed user queries that employ the given API function. The queries exemplify how to utilize the API call. A query should only use the given API. Additionally, you must incorporate the input parameters required for each API call. To achieve this, generate random information for required parameters according to its type. The 5 queries should be very specific. Note that you shouldn't ask 'which API to use', rather, simply state your needs that can be addressed by these APIs. Never explicitly mentioning the specific calling of API functions in your response. You should also avoid asking for the input parameters required by the API call, but instead directly provide the parameters in your query.

Specific requirements:
Please produce 5 queries in line with the given requirements and inputs. These 5 queries should display a diverse range of sentence structures: some queries should be in the form of imperative sentences, others declarative, and yet others interrogative. Equally, they should encompass a variety of tones, with some being polite, some being straightforward, some like layman. Ensure they vary in length. Aim to include a number of engaging queries as long as they relate to API calls. Keep in mind that
- Each query should consist of a minimum of twenty words.
- Never explicitly mentioning the specific calling of API functions in your response. For example, never include API as 'xx.yy.zz(parameters)'.
- The response must be a list of effective json.
- For quotation, use ' .
- Never include the reference paper in your response.
- Never include library keyword or specific type in sentence. Instead, use their description, you can find type description in return type description or parameter type description.
- Restricted to the response format:
[{"Query": "(query content)"},
{"Query": "(query content)"},
{"Query": "(query content)"},
{"Query": "(query content)"},
{"Query": "(query content)"}]

Table A.4: Input and output examples for instruction generation in BioMANIA

# **Example Input**:
Input={
"API": "scanpy.pl.spatial",
"description": "Scatter plot in spatial coordinates. This function allows overlaying data on top of images.",
"required_args": {"adata": {"type": "AnnData", "default": null, "optional": false, "description": "Annotated data matrix.", "optional_value": true}},
"return_schema": "type": "Optional[Union[Axes, List[Axes]]]", "description": "If 'show==False' a : class:' matplotlib.axes.Axes' or a list of it."
}

Example Output:
[{"Query": "Can you overlay gene expression data onto spatial coordinates using the given API function, given an annotated data matrix?"},
{"Query": "Show me a scatter plot in spatial coordinates by overlaying data on top of an image using the given API function."},
{"Query": "Given an annotated data matrix, can the API function create a scatter plot in spatial coordinates with overlaid data on top of an image?"},
{"Query": "How can I visualize gene expression data in spatial coordinates by overlaying it onto an image using the given API function?"},
{"Query": "Please provide a scatter plot in spatial coordinates with overlaid gene expression data on top of an image, using the annotated data matrix as input for the API function."}]

Table A.5: Prompt for API name prediction in BioMANIA

Task Description:
Task: choose one of the following APIs to use for the instruction. {retrieved_apis: their descriptions}

Instruction: {example_query}
API: {example_answer}

Instruction: {user_query}
API:

Table A.6: Input and output examples for API name prediction in BioMANIA

# **Example Input**:
{
"scanpy.pp.subsample": "Subsample to a fraction of the number of observations"
"scanpy.get.obs_df": "Return values for observations in adata"
"scanpy.datasets.blobs": "Gaussian Blobs"
}

[
{"example_query": "Could you please perform PhenoGraph clustering on the provided single-cell data? Use the Jaccard similarity coefficient for clustering and set the minimum cluster size to 50.", "example_answer": "scanpy.external.tl.phenograph" },
{"example_query": "Get a dataframe containing gene annotations for rat from ensembl biomart.", "example_answer": "scanpy.queries.biomart_annotations" },
{"example_query": "I would like to see a violin plot that displays the ranking of genes for all tested comparisons in the annotated data matrix.", "example_answer": "scanpy.pl.rank_genes_groups_violin" },
{"example_query": "Please perform PHATE analysis on the given annotated data matrix and return the PHATE coordinates of the data.", "example_answer": "scanpy.external.tl.phate" },
{"example_query": "Can you show me the ranking of genes for group A and group B in the dataset?", "example_answer": "scanpy.pl.rank_genes_groups" },
]

Example Output:
"scanpy.pp.subsample"

Table A.7: Prompt for API parameter prediction in BioMANIA

Input:
USER QUERY: {user_query}
API DESCRIPTION: {api_description}
API NAME: {api_name}
API PARAMETER INFORMATION: {api_parameter_information}

Task Description:
You are a skilled coder, you have been provided USER QUERY, API DESCRIPTION, API NAME, and API PARAMETER INFORMATION. You need to evaluate whether an parameter has been assigned its designated value in the user's query. Afterwards, if the value is indeed assigned, you should proceed to assign that value to its respective parameter. The parameter should be only from: {parameter_list}

Specific requirements:
The order for assigning value is that you should first use the value you find from the query. If not found, then use the default value. You should only return the parameters with assigned value, no matter using default value or found value from the query. If the parameter has no searched value from query and no default value, do not include it in your response. If you're unable to find any value or validate a match or default value, simply return an empty list.

Restricted to the response format: "parameter_name": parameter name, "value": the value you find. Your answer should be a list of valid json. Do not offer any additional information. The value should be found from USER QUERY instead of other information. Please carefully review the user query and API description to ensure that the extracted values match the intended user meaning.

Table A.8: Input and output examples for API parameter prediction in BioMANIA

# **Example Input**:
user_query= "Can you conduct diffusion maps on the provided data?"
api_description= "Scatter plot in Diffusion Map basis."
api_name= "scanpy.pl.diffmap"
api_parameter_information={
"name": "adata",
"type": "AnnData",
"description": "Annotated data matrix.",
"default_value": None
}

Example Output:
{"parameter_name": "adata", "value": None}

Tool	Category	URL
Biopython	General	https://biopython.org/
scikit-bio	General	http://scikit-bio.org/
Biotite	General	https://www.biotite-python.org/
DEAP	Genomics	https://deap.readthedocs.io/en/master/
ETE Toolkit	Genomics	http://etetoolkit.org/
Scanpy	Transcriptomics	https://scanpy.readthedocs.io/en/stable/index.html
Squidpy	Transcriptomics	https://squidpy.readthedocs.io/en/stable/index.html
scvi-tool	Transcriptomics	https://docs.scvi-tools.org/en/stable/
SCENIC+	Transcriptomics	https://scenicplus.readthedocs.io/en/latest/
QIIME 2	Microbiomics	https://docs.qiime2.org/2023.9/
Pyteomics	Proteomics	https://pyteomics.readthedocs.io/en/latest/
pyOpenMS	Proteomics	https://pyopenms.readthedocs.io/en/latest/

Table A.9: Well-documented open-source Python tools investigated by BioMANIA.

API	Synthetic instructions
scanpy.pp.pca	1. Perform principal component analysis on the given data matrix to obtain PCA coordinates, loadings, and variance decomposition. 2. Can you calculate the PCA representation of the data and provide the principal components containing the loadings? 3. I need the PCA coordinates and variance ratio for the given data matrix. Can you help? 4. Compute the PCA representation of the data and return the explained variance, which is equivalent to the eigenvalues of the covariance matrix. 5. Please run PCA on the data matrix and return the PCA representation along with the explained variance and variance ratio.
scanpy.pl.embedding	1. Create a scatter plot using the 'umap' basis and color the points based on a specific gene expression level. 2. Generate a scatter plot using the 'pca' basis and highlight the edges between neighboring points. 3. Plot a scatter plot using the 'tsne' basis and adjust the size of the points based on a specific gene expression level. 4. Display a scatter plot using the 'diffmap' basis and use a specific color map to represent the points. 5. Visualize a scatter plot using the 'umap' basis and add a color bar to indicate the range of expression levels.
scanpy.external.pp.mnn_correct	1. Correct batch effects in expression matrices using mutual nearest neighbors matching with MNN correction API. 2. Can you please correct the batch effects in my AnnData object using the MNN correction API? 3. Apply MNN correction to my expression matrices to remove batch effects and obtain corrected matrices. 4. I need to correct the batch effects in my expression matrices. Can you suggest an API that can perform this task? 5. Is there an API available that can correct batch effects in expression matrices by matching mutual nearest neighbors?
scanpy.pp.regress_out	1. Can you please regress out the unwanted sources of variation using simple linear regression on the annotated data matrix? 2. I need to remove the unwanted sources of variation from the data matrix. Can you help me with that? 3. Could you perform linear regression to eliminate the unwanted sources of variation from the annotated data matrix? 4. Please correct the data matrix by regressing out the unwanted sources of variation using linear regression. Is it possible to use linear regression to remove the unwanted sources of variation from the annotated data matrix?

Table A.10: Synthetic instruction examples of random APIs generated by BioMANIA.

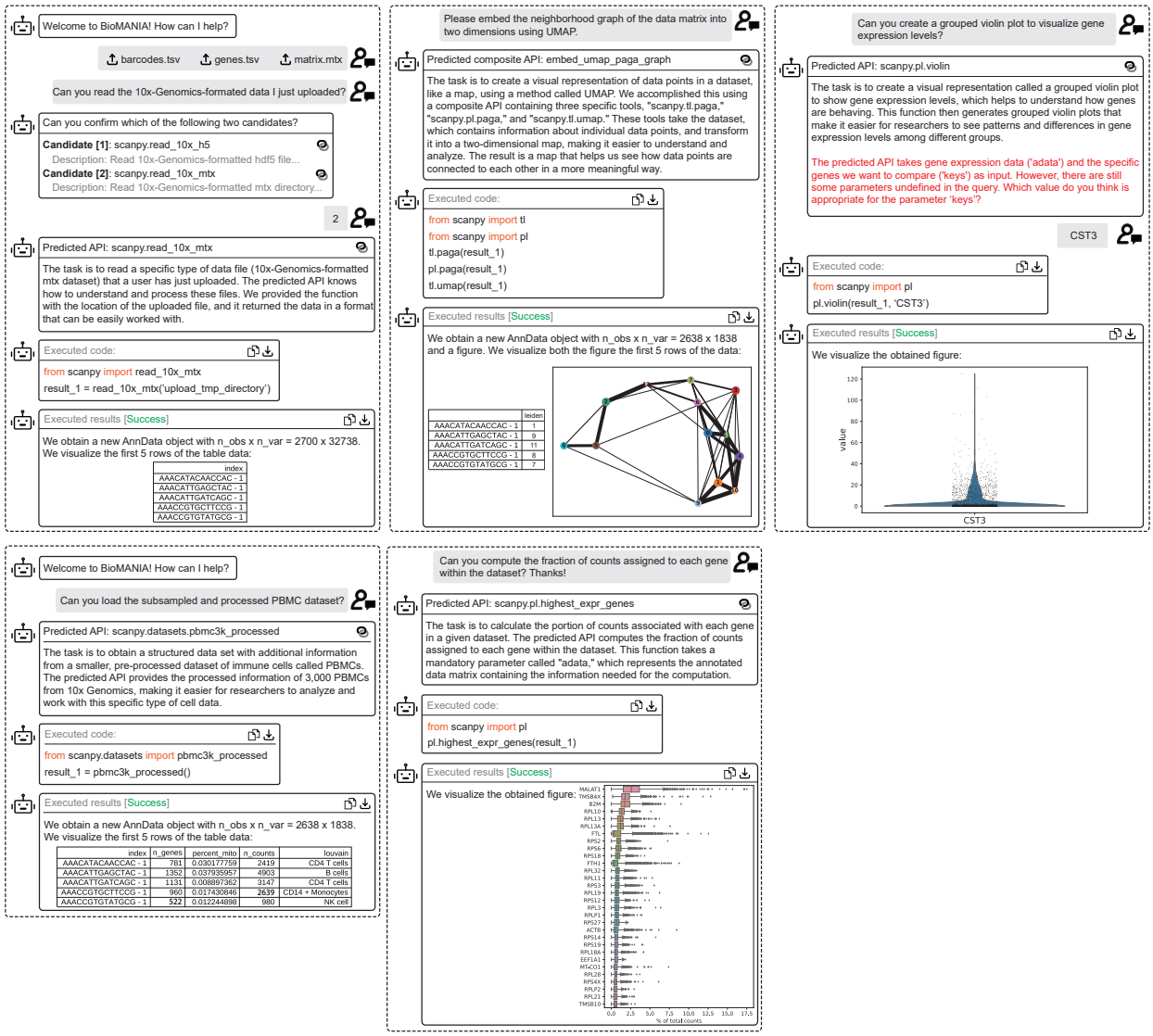

Figure A.1: **More illustrative examples of using the Scanpy ChatBot for single-cell data analysis.**