Journal of the American Statistical Association

ISSN: 0162-1459 (Print) 1537-274X (Online) Journal homepage: www.tandfonline.com/journals/uasa20

# LAMBDA: A Large Model Based Data Agent

Sun Maojun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan & Jian Huang

**To cite this article:** Sun Maojun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan & Jian Huang (02 Jun 2025): LAMBDA: A Large Model Based Data Agent, Journal of the American Statistical Association, DOI: 10.1080/01621459.2025.2510000

**To link to this article:** https://doi.org/10.1080/01621459.2025.2510000

# LAMBDA: A Large Model Based Data Agent

Maojun Sun[b], Ruijian Han[b], Binyan Jiang[b], Houduo Qi[a,b], Defeng Sun[a], Yancheng Yuan[a,*] and Jian Huang[a,b,*]

[a]Department of Applied Mathematics, The Hong Kong Polytechnic University

[b]Department of Data Science and Artificial Intelligence, The Hong Kong Polytechnic University

*Corresponding authors: j.huang@polyu.edu.hk, yancheng.yuan@polyu.edu.hk



## Abstract

We introduce LArge Model Based Data Agent (LAMBDA), a novel open-source, code-free multi-agent data analysis system that leverages the power of large language models. LAMBDA is designed to address data analysis challenges in data-driven applications through innovatively designed data agents using natural language. At the core of LAMBDA are two key agent roles: the programmer and the inspector, which are engineered to work together seamlessly. Specifically, the programmer generates code based on the user's instructions and domain-specific knowledge, while the inspector debugs the code when necessary. To ensure robustness and handle adverse scenarios, LAMBDA features a user interface that allows direct user intervention. Moreover, LAMBDA can flexibly integrate external models and algorithms through our proposed Knowledge Integration Mechanism, catering to the needs of customized data analysis. LAMBDA has demonstrated strong performance on various data analysis tasks. It has the potential to enhance data analysis paradigms by seamlessly integrating human and artificial intelligence, making it more accessible, effective, and efficient for users from diverse backgrounds. The strong performance of LAMBDA in solving data analysis problems is demonstrated using real-world data examples. The code for LAMBDA is available at https://github.com/AMA-CMFAI/LAMBDA and videos of three case studies can be viewed at https://www.polyu.edu.hk/ama/cmfai/lambda.html.

*Keywords:* Code generation via natural language; Data analysis; Large models; Multi-agent collaboration; Software system.

# 1 Introduction

Over the past decade, the data-driven approach utilizing deep neural networks has driven the success of artificial intelligence across many challenging applications in various fields (LeCun et al., 2015). Despite these advancements, the current paradigm encounters challenges and limitations in statistical and data science applications, particularly in domains such as biology (Weissgerber et al., 2016), healthcare (Oakes et al., 2024), and business (Weihs and Ickstadt, 2018), which require extensive expertise and advanced coding knowledge for data analysis. A significant barrier is the lack of effective communication channels between domain experts and sophisticated AI models (Park et al., 2021). To address this issue, we introduce a Large Model Based Data Agent (LAMBDA), which is a new open-source, code-free multi-agent data analysis system designed to overcome this dilemma. LAMBDA aims to create a much-needed medium, fostering seamless interaction between domain knowledge and the capabilities of AI in statistics and data science.

Our main objectives in developing LAMBDA are as follows.

**(a) Crossing coding barrier**: Coding has long been recognized as a significant barrier for domain experts without a background in statistics or computer science, preventing them from effectively leveraging powerful AI tools for data analysis (Oakes et al., 2024). LAMBDA addresses this challenge by enabling users to interact with data agents through natural language instructions, thereby offering a coding-free experience. This approach significantly lowers the barriers to entry for tasks in data science, such as data analysis and data mining, while simultaneously enhancing efficiency and making these tasks more accessible to professionals across various disciplines.

**(b) Integrating human intelligence and AI**: The existing paradigm of data analysis is confronted with a challenge due to the lack of an efficient intermediary that connects human intelligence with artificial intelligence (Park et al., 2021). On one hand, AI models often lack an understanding of the unlearned domain knowledge required for specific tasks. On the other hand, domain experts find it challenging to integrate their expertise into AI models to enhance their performance (Dash et al., 2022). LAMBDA provides a possible solution to alleviate this problem. With a well-designed interface in our key-value (KV) knowledge base, the agents can access external resources like algorithms or models. This integration ensures that domain-specific knowledge is effectively incorporated, meets the need for customized data analysis, and enhances the agent's ability to perform complex tasks with higher accuracy and relevance.

**(c) Reshaping data science education**: LAMBDA has the potential to become an interactive platform that can transform statistical and data science education. It offers educators the flexibility to tailor their teaching plans and seamlessly integrate the latest research findings. This adaptability makes LAMBDA an invaluable tool for educators seeking to provide cutting-edge, personalized learning experiences. Such an approach stands in contrast to the direct application of models like GPT-4 (OpenAI, 2023; Tu et al., 2024), offering a unique and innovative educational platform.

Beyond these features, the design of LAMBDA also emphasizes reliability and portability. Reliability refers to LAMBDA's ability to handle data analysis tasks stably and automatically address failures. Portability ensures that LAMBDA is compatible with various large language models (LLMs), allowing it to be continuously enhanced by the latest state-of-the-art models. To save users time on tasks such as document writing, LAMBDA is equipped with the capability for automatic analysis report generation. To accommodate diverse user needs, LAMBDA also supports exporting code to IPython notebook files, such as "ipynb" files in Jupyter Notebook.

While GPT-4 has demonstrated state-of-the-art performance in advanced data analysis, its closed-source nature constrains its adaptability to the rapidly expanding needs of statistical and data science applications and specialized educational fields. Furthermore, concerns regarding data privacy and security risks are inherent in the present configuration of GPT-4 (Bavli et al., 2024). In contrast, by utilizing the open-source LAMBDA, users can alleviate concerns about data privacy by preventing the transmission of user data to external servers. Additionally, it offers greater flexibility and convenience in integrating domain knowledge, installing packages, and utilizing various computational resources.

LAMBDA demonstrates exceptional performance across various datasets used in our system testing. Moreover, it outperforms other data agents in handling complex domain tasks during our experiments. In summary, our main contributions are as follows: We propose a well-engineered architecture for an LLM-based data agent that enables natural language-driven data analysis in a conversational manner. Unlike typical end-to-end data agents, our design allows human intervention throughout the process, ensuring adaptability when the agent fails to complete a task or misinterprets user intent. Moreover, we introduce a Knowledge Integration mechanism to effectively handle tasks requiring domain-specific knowledge, providing greater flexibility when misalignment occurs in the knowledge. Its ongoing development has the potential to enhance statistics and data science, making advanced tools more accessible to diverse users.

This paper begins with the background and related works in Section 2. Section 3 provides a detailed description of the proposed LAMBDA method. To evaluate its effectiveness, we present our experiments and results in Section 4. Section 5 demonstrates examples and cases of LAMBDA's application in various scenarios, including data analysis, integration of human intelligence, and interactive education. The paper concludes with a summary in Section 6. More information and details, including implementation, some discussions, datasets, case studies, and experimental settings, are provided in the Supplementary Materials.

## 2 Background and related works

In recent years, the rapid progress in LLMs like GPT-3, GPT-4, PaLM, LLaMA, and Qwen (Brown et al., 2020; OpenAI, 2023; Chowdhery et al., 2022; Touvron et al., 2023; Bai et al., 2023) has brought boundless possibilities to the field of artificial intelligence and its applications in many fields, including statistics and data science. Benefiting from this revolution, LLM-powered agents (LLM agents) are developed to automatically solve problems in various domains like the search engine, software engineering, gaming, and data science (Guo et al., 2024; Hong et al., 2023; Wu et al., 2023; Zhou et al., 2023; Hong et al., 2023).

## 2.1 LLMs as data analysis agents

LLM-based data science agent, or data agent, is dedicated to harnessing the power of LLMs to automate data science and analysis tasks (Sun et al., 2024). For example, GPT-4-Advanced Data Analysis and ChatGLM-Data Analysis can analyze user's data files, perform computations, and generate visualizations (OpenAI, 2023). Some works integrate LLMs into Jupyter Notebooks. For instance, MLCopilot (Zhang et al., 2023) and Chapter (Chapyter, 2023), enable users to interact directly with the notebook, greatly enhancing flexibility. However, they cannot automatically fix errors when they occur and require additional magic commands to support natural language input.

Meanwhile, some researchers focus on designing end-to-end data agents to automate the entire pipeline, including data preprocessing and model evaluation, without human intervention. For example, Data Interpreter (Hong et al., 2024) and TaskWeaver (Qiao et al., 2023) accomplish their tasks through planning and iterative steps. However, current state-of-the-art LLM/VLM-based agents do not reliably automate complete data science workflows (Cao et al., 2024). While fully relying on LLMs for each step reduces human effort, it also significantly increases instability and uncertainty. In addition, if any intermediate step does not align with the user's intent, the process must be repeated, potentially leading to token waste. In contrast, LAMBDA is designed to support a human-agent collaboration mode, allowing for human intervention at any stage of the process if necessary.

Furthermore, these works have not adequately addressed the high degree of user flexibility needed in data analysis, such as the integration of custom algorithms or statistical models according to user preferences. This flexibility is crucial for enhancing data analysis tasks in domain-specific applications and in statistical and data science education. To address this gap, we have designed a Knowledge Integration Mechanism that allows for the easy incorporation of user resources into our agent system.

## 2.2 Multi-agent collaboration

A multi-agent system consists of numerous autonomous agents that collaboratively engage in planning, discussions, and decision-making, mirroring the cooperative nature of human group work in problem-solving tasks (Guo et al., 2024). Each agent has unique capabilities, objectives, and perceptions, operating either independently or collectively to tackle complex tasks or resolve problems (Huang et al., 2023a). Hong et al. (2023) proposed MetaGPT, modeled after a software company, consisting of agents such as Product Manager, Architect, Project Manager, Engineer, and QA Engineer, efficiently breaking down complex tasks into subtasks involving many agents working together. However, even for simple tasks like data visualization, MetaGPT consume a large number of tokens and require more time. In addition, they generate engineering files that need manual execution and lack the immediacy and interactivity essential for intuitive data analysis. In contrast, LAMBDA simplifies the collaboration process by involving only two agents to simulates data analysis workflows, programmer and inspector respectively, reducing token and time consumption. Moreover, its well-designed user interface allows users to intuitively view the analysis results directly on the screen. A comparison and discussion can be found in the supplement materials.

## 2.3 Knowledge integration

Addressing tasks that require domain-specific knowledge presents a significant challenge for AI agents (Zhang et al., 2024). Incorporating knowledge into LLMs through in-context learning (ICL) is a promising strategy for acquiring new information. A well-known technique in this regard is retrieval-augmented generation (RAG) (Gao et al., 2023), which enhances the accuracy and reduces hallucinations of LLM answers by retrieving external sources (Lewis et al., 2020; Huang et al., 2023b; Borgeaud et al., 2022; Mialon et al., 2023). In RAG, resources are divided into sub-fragments, embedded into vectors, and stored in a vector database. The model first queries this database, identifying document fragments relevant to the user's query based on the similarity. These fragments are then utilized to refine the answers generated by the LLMs through ICL (Lewis et al., 2020). However, deploying a general RAG approach in data analysis introduces specific challenges. First, the user's instructions may not align closely with the relevant code fragments in the representation space, resulting in inaccurate searches. Second, when dealing with extensive code, the agents might struggle to contextualize the correct code segments, where accuracy and completeness are essential for codes and final results.

In addition, custom APIs (Hong et al., 2024) can be implemented to handle domain-specific tasks (Qiao et al., 2023; Hong et al., 2024). For example, systems like Data Interpreter and TaskWeaver invoke the corresponding Tools/Plugins directly within the generated code. Compared to direct parameter-passing, this approach offers greater flexibility in tool usage. However, since the agent cannot access the implementation details of these plugins, it is limited to simple plugin usage and may struggle to resolve misalignment between tools and human instructions when plugin usage is inappropriate.

To address these challenges, we develop a specially designed KV knowledge base with integration methods. This allows users to choose between different modes, including 'Full' and 'Core', based on the complexity, length of the knowledge context, and specific task requirements. By integrating knowledge, our agent system becomes more adaptable to domain-specific tasks, leveraging human expertise more effectively.

# 3 Methodology

Our proposed multi-agent data analysis system, LAMBDA, consists of two agents that cooperate seamlessly to solve data analysis tasks using natural language, as shown in Figure 1. The macro workflow describes the code generation process based on user instructions and subsequently executing that code.

## 3.1 Overview

LAMBDA is structured around two core agent roles: the "programmer" and the "inspector," who are tasked with code generation and error evaluation, respectively. The two agents can be implemented separately using either the same or different LLMs. When users submit an instruction, the programmer agent writes code based on the provided instruction and dataset. This code is then executed within the kernel of the host system. Should any errors arise during execution, the inspector intervenes, offering suggestions for code refinement. The programmer takes these suggestions into account, revises the code, and resubmits it for re-evaluation. This iterative cycle continues until the code runs error-free or a preset maximum

number of attempts is reached. In order to cope with adverse situations and enhance its reliability and flexibility, a human intervention mechanism is integrated into the workflow. This feature allows users to modify and run the code directly and intervene when necessary. The multi-agent collaboration algorithm is demonstrated in Algorithm 1.

**Algorithm 1** Multi-agent Collaboration. $A_n$, $C_n$ are the answer and extracted code by the programmer agent in iteration *n*. We assume each $A_n$ contains $C_n$, otherwise, the programmer's reply will be returned to the user directly. *r* is the execution result, *E* indicates an error, $S_n$ are suggestions provided by the inspector in iteration *n*, $C_h$ is the code written by a human. The final response is denoted as *R*.

**Require:** *Pr*: Programmer agent

**Require:** *I*: Inspector agent

**Require:** *d*: Dataset provided by user

**Require:** *ins*: Instructions provided by user

**Require:** *T*: Maximum number of attempts

1: $n \leftarrow 0$ ▷ Initialize iteration counter

2: $C_n \leftarrow A_n,\ A_n \leftarrow Pr(d, ins)$ ▷ Extract code and answer by Programmer

3: $r = \begin{cases} r, & \text{success} \\ E, & \text{error} \end{cases} \leftarrow \text{execute}(C_n)$ ▷ Code execution, similarly to subsequent *r*

4: **while** $r = E$ **and** $n < T$ **do** ▷ Self-correcting mechanism start

5: $n \leftarrow n+1$

6: $S_n \leftarrow I(C_{n-1}, E)$ ▷ Inspector provides suggestions

7: $C_n \leftarrow A_n, A_n \leftarrow Pr(C_{n-1}, S_n, E)$ ▷ Programmer modifies code

8: $r \leftarrow \text{execute}(C_n)$ ▷ Execute modified code

9: **end while**

10: **if** $r = E$ **then**

11: $r \leftarrow \text{execute}(C_h)$ ▷ Human intervention (Optional)

12: $R \leftarrow C_h \cup Pr(r)$ ▷ Final response in natural language

13: **end if**

14: $R \leftarrow C_n \cup Pr(r)$ ▷ Final response in natural language

## 3.2 Programmer agent

The main responsibility of the programmer is to write code and respond to the user. Upon the user's dataset upload, the programmer receives a tailored system prompt that outlines the programmer's role, environmental context, and the I/O formats. This prompt is augmented with examples to facilitate few-shot learning for the programmer. Specifically, the system prompt encompasses the user's working directory, the storage path of the dataset, the dimensions of the dataset, the name of each column, the type of each column, information on missing values, and statistical description.

The programmer's workflow can be summarized as follows: initially, the programmer writes code based on instructions from the user or the inspector; subsequently, the program extracts code blocks from the programmer's output and executes them in the kernel. Finally, the programmer generates a final response based on the execution results and communicates it to the user. This final response consists of a summary and suggestions for the next steps.

## 3.3 Inspector agent and self-correcting mechanism

The inspector's role is to provide modification suggestions when errors occur in code execution. The prompt of the inspector includes the code written by the programmer during the current dialogue round and the error messages from the kernel. The inspector will offer actionable revision suggestions to the programmer for code correction. This suggestion prompt contains the erroneous code, kernel error messages, and the inspector's suggestions. This collaborative process between the two agents iterates several rounds until the code executes successfully or the maximum number of attempts is reached. This self-correcting mechanism enables the programmer and inspector to make multiple attempts in case of error. A case of self-correcting mechanism and released experiment can be found in the Supplementary Materials.

## 3.4 Integrating human intelligence and AI

Beyond leveraging the inherent knowledge of LLMs, LAMBDA is further enhanced to integrate human intelligence through external resources such as customized algorithms and statistical models from users. As mentioned above, the challenges faced by general RAG methods in data analysis stem from the potential lack of clear correlation between user instructions and code fragments in the representation space, as well as the impact of the length of code fragments. We design a special KV knowledge base for this challenge.

The KV knowledge base is a repository for housing external resources from users in key and value pairs. Specifically, we format the code of resources into key-value pairs: the key represents the resource description, and the value denotes the code. The user's query will be matched within the knowledge base to select the code with the highest similarity. Figure 2 demonstrates the workflow of knowledge matching in LAMBDA. We define the knowledge base as $\mathcal{K} = \{(d_i, c_i) | \ i = 1, 2, \ldots, n\}$, where $d_i$ represents the description of the *i*-th piece of knowledge and $c_i$ represents the corresponding source code.

When the user issues an instruction *ins*, an embedding model $\mathcal{F}$ encodes all descriptions in the knowledge base and the *ins*, such as Sentence-BERT (Reimers and Gurevych, 2019). The

embedding tensors for descriptions and instruction are represented by $\mathbf{e}_{d_i}$ and $\mathbf{e}_{ins}$ respectively. The cosine similarity between them is calculated to select knowledge with a similarity score greater than a threshold $\theta$, with the highest-scoring match chosen as the relevant knowledge.

Let the embedding function be $\mathcal{F}$, the $\mathbf{e}_{d_i}$ and $\mathbf{e}_{ins}$ are formulated as follows $\mathbf{e}_{d_i} = \mathcal{F}(d_i), i \in \{1, 2, \ldots, n\}$, and $\mathbf{e}_{ins} = \mathcal{F}(ins)$. The similarity $S_i$ between description and instruction is computed using cosine similarity as

$$S_i(\mathbf{e}_{d_i}, \mathbf{e}_{ins}) = \frac{\mathbf{e}_{d_i} \cdot \mathbf{e}_{ins}}{\| \mathbf{e}_{d_i} \| \| \mathbf{e}_{ins} \|} \quad \forall i \in \{1, 2, \ldots, n\}.$$

The matched knowledge *k* with the highest $S_i$ is selected while it satisfies $S_i > \theta$, computed as

$$k = c_{i^*}, \quad i^* = \arg\max_i \left( S_i(\mathbf{e}_{d_i}, \mathbf{e}_{ins}) \cdot \mathbf{1}_{\{S_i(\mathbf{e}_{d_i}, \mathbf{e}_{ins}) > \theta\}} \right) \quad \forall i \in \{1, 2, \ldots, n\}.$$

The knowledge *k* will be embedded in ICL for the LLM to generate answer $\hat{A}$. Formally, given a query *q*, matched knowledge *k*, a set of demonstrations $D = \{(q_1, k_1, a_1), (q_2, k_2, a_2), \ldots, (q_n, k_n, a_n)\}$, and the LLM $\mathcal{M}$, the model estimates the probability $\mathcal{P}(a \mid q, k, D)$ and outputs the answer $\hat{A}$ that maximizes this probability. The final response $\hat{A}$ is generated by the model $\mathcal{M}$ as $\hat{A} \leftarrow \mathcal{M}(q, D)$.

The matching threshold $\theta$ defines the required similarity between a knowledge description and a user instruction, directly influencing the complexity of retrieving relevant knowledge. A higher $\theta$ imposes stricter matching criteria, reducing the chance of retrieval, whereas a lower $\theta$ increases the probability of identifying a match.

The optimal selection of $\theta$ depends on multiple factors. For example, when users aim to incorporate specific knowledge into a task, a lower $\theta$ value increases the chance of retrieving the relevant information. Furthermore, the length of the knowledge description plays a critical role, as longer descriptions typically necessitate a lower $\theta$ value since user instructions are generally more concise. By default, we recommend setting $\theta$ to 0.2. However, this value can be adjusted based on the aforementioned factors to optimize retrieval performance.

By integrating *k* through ICL, the model effectively combines retrieved domain knowledge with contextual learning to provide answers that are more accurate. Moreover, LAMBDA offers two integration modes: ‘Full’ and ‘Core’. In the ‘Full’ mode, the entire knowledge is utilized as the context in ICL. In the ‘Core’ mode, the core functions are processed through ICL, while other functions are executed directly in the back-end. This approach allows the agents to focus on modifying the core function directly, without the need to understand or implement the sub-functions within it. The ‘Core’ mode is particularly effective for scenarios involving lengthy code, as it eliminates the need to process the entire code through ICL. These two modes of knowledge integration provide substantial flexibility for handling tasks

that require domain-specific knowledge. We evaluate our Knowledge Integration Mechanism in Table 8 through several domain tasks.

In summary, the Knowledge Integration Mechanism empowers LAMBDA to perform domain tasks and offers the flexibility needed to address complex data analysis challenges.

## 3.5 Kernel, report generation and code exporting

LAMBDA uses IPython as its kernel to manage sequential data processing, where each operation builds on the previous one, such as standardization followed by one-hot encoding. Implementation details are in the Supplementary Materials. LAMBDA also generates analysis reports from dialogue history, including data processing steps, visualizations, model descriptions, and evaluation results. Users can choose from various report templates, and the agent creates reports via ICL, allowing users to focus on higher-value tasks. A sample report is in Figure 9 and the Supplementary Materials. Moreover, users can download their experimental code as an IPython notebook.

## 3.6 User interface

LAMBDA provides an accessible user experience similar to ChatGPT. Users can upload datasets and describe tasks in natural language, supported by LLMs like Qwen-2, which recognizes 27 languages. It is recommended to prompt LAMBDA step-by-step, mimicking data analysts' approach, to maintain control and embody the "human-in-the-loop" concept. LAMBDA generates results, including code, figures, and models, which users can copy and save with a single click. Even those without expertise in statistics or data science can train advanced models by simply asking for recommendations, such as XGBoost and AdaBoost. Advanced users can customize LAMBDA's knowledge via an interface template. Users can also export text reports and code for further study. A usage example is shown in Figure 9. LAMBDA's interface is designed to be accessible to users of all backgrounds.

To summarize, the programmer agent, inspector agent, self-correcting mechanism, and human-in-the-loop components collectively ensure the reliability of LAMBDA. The integration of knowledge makes LAMBDA scalable and flexible for domain-specific tasks. To enhance portability, we provide an OpenAI-style interface for LAMBDA. This ensures that most LLMs, once deployed via open-source frameworks such as vLLM (Kwon et al., 2023) and LLaMA-Factory (Zheng et al., 2024b), are compatible with LAMBDA.

## 3.7 Prompt

We present examples of prompts for the roles of programmer, inspector, self-corrector, and knowledge integrator. Additional prompt examples and case studies are available in the Supplementary Materials.

Figure 3 gives an example prompt for the data analyst at the start of the analysis session.

Figure 4 shows a system prompt about the dataset, which provides essential information to the programmer agent.

After obtaining the execution results, a prompt such as the one given in Figure 5 can be used to format the output, enabling the programmer agent to provide an explanation or suggest the next steps.

When an error occurs, a prompt for the inspector is employed to guide the inspector in identifying the cause of the bug and to offer revision suggestions (Figure 6).

Figure 7 presents an example prompt for the programmer revising the error code.

For knowledge integration, the system message prompt and retrieval result are shown in Figure 8.

# 4 Experiments

## 4.1 Data experiments

The current data analysis paradigm relies on programming software and languages such as R (R Core Team, 2023), SAS (SAS Institute Inc., 2015), and Python (Python Software Foundation, 2023) for computation and experimentation. To gain practical experience and evaluate LAMBDA's performance in real-world data science tasks, we first applied LAMBDA to several standard datasets for classification and regression analysis. In addition, we conducted further investigations in broader statistical analysis scenarios, such as high-dimensional data, missing data, image data, and text data, to examined its robustness and versatility. All information of the datasets used can be found in the supplementary materials.

For classification problems, we measured accuracy on the test data, defined as the ratio of correctly classified instances to the total number of instances. For regression problems, we used Mean Squared Error (MSE), which is the average of the squared differences between the predicted values and the actual values in the test data. The formula for MSE is:

$\mathrm{MSE} = (1/n)\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, where $n$ is number of data points, $y_i$ is the observed value, $\hat{y}_i$ is the predicted value. We employed 5-fold cross validation for evaluation in all the cases. Table 1 lists the datasets used in our experiments and case studies.

### 4.1.1 Experiments with classical tabular data

We initially applied LAMBDA to several classical datasets, covering both classification and regression tasks. To facilitate comparison, we documented the analysis methods employed by LAMBDA and then manually conducted the same analyses using R. The results are summarized in Table 2, with the corresponding results from the R analyses presented in parentheses.

The results presented in Table 2 demonstrate LAMBDA's robust performance in executing data analysis tasks. These results are either superior to or on par with those obtained using R. These outcomes highlight LAMBDA's effectiveness in leveraging various models across tabular data scenarios. Furthermore, the results indicate that LAMBDA performs at a level comparable to that of a data analyst proficient in R. This suggests the potential for systems like LAMBDA to become indispensable tools for data analysis in the future. Notably, there

was no human involvement in the entire experimental process with LAMBDA, as only prompts in English were provided.

In summary, the experimental results demonstrate that LAMBDA achieves human-level performance and can serve as an efficient and reliable data agent, assisting individuals in handling data analysis tasks.

### 4.1.2 Experiments with high-dimensional data and unstructured data

To validate LAMBDA's robustness and versatility, we further explored its application across a broader range of data scenarios, including high-dimensional data, missing data, image data, and text data.

• **High-dimensional data:** We evaluated LAMBDA on the following three challenging high-dimensional clinical datasets: TCGAmirna (Bentink et al., 2012), EMTAB386 (Colaprico et al., 2015), and GSE49997 (Pils et al., 2012).

We summarize the sample size and dimensions in Table 3. The test results are presented in Table 4. More detailed descriptions of these three datasets are given in the Supplementary Materials. We found that LAMBDA consistently applies dimensionality reduction techniques, such as Principal Component Analysis (PCA), as a preprocessing step. This allows us to apply methods like logistic regression without the regularization. The results indicate that LAMBDA is capable of handling high-dimensional data.

• **Missing data:** We evaluated LAMBDA on three datasets containing missing values, with results summarized in Table 5. We observe that LAMBDA tends to prioritize deleting the observations that contain missing values. However, with an appropriate prompt, LAMBDA can also attempt to impute missing values (e.g., mean value). When errors arise due to missing values, the Inspector agent effectively identifies the issue, notifies the Programmer agent, and applies the necessary corrections.

• **Image data:** We used LAMBDA to train a handwritten digit classifier based on the MNIST dataset. We prompted LAMBDA to utilize various neural network architectures, such as Convolutional Neural Networks (CNNs) and Transformers, as backbone models. The results of this experiment are presented in Table 6. According to Table 6, we find LAMBDA can effectively implement and apply deep learning architectures like CNNs and Transformers for image classification tasks.

• **Text data:** We used LAMBDA to train a spam detection classifier based on the SMS Spam Collection Dataset. Similar to our approach with image data, we prompted LAMBDA to experiment with different backbone models for this task. The results are summarized in Table 7. As shown in Table 7, LAMBDA successfully performed text classification tasks. Notably, when prompted to use a Transformer-based architecture, LAMBDA employed DistilBERT-Base-Uncased for transfer learning, which significantly improved both training efficiency and model performance.

Overall, our findings indicate that LAMBDA is not only capable of handling tabular tabular tasks but also effectively processing image and text data. In future work, we aim to explore more complex and diverse data scenarios.

## 4.2 Performance of Knowledge Integration

We collected three domain-specific tasks to evaluate the proposed Knowledge Integration Mechanism and compare it with advanced data analysis agents. Specifically, the tasks involve utilizing the recent algorithm packages (e.g., PAMI (Piotrowski et al., 2021)), implementing optimization algorithms (e.g., computing the nearest correlation matrix), and training the latest research models (e.g., non-negative neural networks). For each task, we define a score $\mathcal{S}$ that is calculated as follows:

$$\mathcal{S} = \begin{cases} 0, & \text{code error and execution error, or exceeded runtime limit,} \\ 0.5, & \text{code error and execution successful,} \\ 0.8, & \text{code successful, execution error due to other issues, e .g. environment,} \\ 1, & \text{both code and execution successful.} \end{cases}$$

To ensure maximum alignment in experimental settings, we converted the code into corresponding tools for agents equipped with a tools mechanism. For agents lacking such a mechanism, we directly included the code in their context. All agents are implemented using GPT-3.5, except for methods and platforms that have their own models, such as GPT-4-Advanced Data Analysis, ChatGLM-Data Analysis, and OpenCodeInterpreter. Since each task can be completed within one minute, we set a maximum runtime limit of 5 minutes to prevent some agents from becoming stuck in infinite self-modification loops.

• *Pattern Mining* Piotrowski et al. (2021) introduce PAMI (PAttern MIning), a cross-platform, open-source Python library offering algorithms to uncover patterns in diverse databases across multiple computing architectures.

• *Nearest Correlation Matrix* Qi and Sun (2006) propose a Newton-type method specifically designed for the nearest correlation matrix problem. Numerical experiments validate the method's fast convergence and high efficiency.

• *Fixed Points Non-negative Neural Networks* Rage et al. (2024) analyze nonnegative neural networks, which are defined as neural networks that map nonnegative vectors to nonnegative vectors.

Table 8 demonstrates the effectiveness of LAMBDA's Knowledge Integration mechanism. Specifically, our results showed that many methods scored zero, particularly when the code was lengthy or involved unfamiliar packages not encountered during LLM training. In these situations, most other approaches struggle with one-shot learning. Two exceptions are Data Interpreter and TaskWeaver, which successfully complete the task using pre-defined Plugins/Tools. With the pre-defined Plugins/Tools, they can execute operations internally without requiring the LLM to generate precise code. This mechanism is similar to the 'Core' mode of our LAMBDA.

With these tools, the LLM only needs to learn a given code usage example rather than generating the full internal implementation, even when it has access to those details. Although these approaches are generally suitable, the agent is likely to make mistakes when there is the certain misalignment between the users' instructions and integrated knowledge. In

such circumstances, we need to utilize the 'Full' mode of our LAMBDA. To further support our claim, we designed two additional experiments.

We take the fixed point non-negative neural networks as a example. We further explore the following two cases that involve misalignment in integrating knowledge/tools and human instruction, which require modifications to the tools (the loss and network mapping are annotated in the schema):

• **Case 1:** The instruction specifies the use of L1 Loss, whereas the tool are originally configured with MSE Loss.

• **Case 2:** The instruction specifies a network structure mapping as follows:

– Encoder: $784 \rightarrow 400$, whereas $784 \rightarrow 200$ originally configured.

– Decoder: $400 \rightarrow 784$, whereas $200 \rightarrow 784$ originally configured.

From Table 9, we observe that in Cases 1 and 2, which require modifications to the tools, both TaskWeaver and Data Interpreter directly use the original tools without recognizing that the tools no longer meet the new requirements although the loss and network mapping are annotated in the schema. In contrast, due to the visibility of the knowledge code under 'Full' mode, LAMBDA identifies that the original code cannot satisfy the new requirements, makes the necessary adjustments, and successfully completes the two cases.

# 5 Examples

We present an example of using LAMBDA for building a classification model in Figure 9. We also provide three case studies in video format to demonstrate the use of LAMBDA in data analysis, integrating human intelligence and AI, and education.

• *Data Analysis* We simulate scenarios in which the user requests LAMBDA to perform various tasks, including data preprocessing, data visualization, and model training, on the provided Iris dataset (Fisher, 1988). LAMBDA consistently delivers accurate responses. Additionally, LAMBDA generates an analysis report based on the chat history. A demonstration of this process is given in the first video at https://www.polyu.edu.hk/ama/cmfai/lambda.html.

• *Integrating Human Intelligence and AI* We demonstrated the Knowledge Integration capabilities of LAMBDA by computing the nearest correlation matrix using the Quadratically Convergent Newton Method. We first highlighted the limitations of GPT-4-Advanced Data Analysis in performing this task, thereby underscoring the value of LAMBDA through comparison. A demonstration is given in the second video at https://www.polyu.edu.hk/ama/cmfai/lambda.html.

• *Interactive Education* We consider an educational scenario in which the teacher uses LAMBDA to design the exercise assignments, and the students use LAMBDA to complete exercises. The exercise dataset used is Abalone. This educational support system enhances the efficiency of both teaching and learning. A demonstration is given in the third video at https://www.polyu.edu.hk/ama/cmfai/lambda.html.

# 6 Conclusion

LAMBDA is an open-source multi-agent data analysis system that effectively integrates human intelligence with artificial intelligence. Experimental results demonstrate that LAMBDA achieves satisfactory performance in handling various data analysis tasks. In the future, LAMBDA can be further enhanced with advanced planning, reasoning techniques, and knowledge integration methods to address a broader range of domain-specific tasks. Our results and examples underscore the significant potential of LAMBDA to enhance both statistical and data science practice and education.

By bridging the gap between human expertise and AI capabilities, LAMBDA aims to democratize data science and statistical analysis, fostering a more inclusive environment for innovation and discovery. Its open-source nature encourages collaboration and continuous improvement from the global research community, allowing researchers and developers to contribute to its evolution. As LAMBDA continues to develop, it has the potential to become an invaluable tool for statisticians, data scientists, and domain experts, enhancing their ability to analyze data efficiently and effectively.

Moreover, LAMBDA holds significant potential for statistical and data science education. Its natural language interface lowers barriers for educators and students, enabling them to focus on problem formulation rather than getting bogged down by syntactic complexities. By generating executable code for various tasks, LAMBDA provides immediate, actionable feedback, which can enhance the learning experience by allowing students to see the direct impact of their queries and hypotheses. This capability not only aids in teaching fundamental concepts but also empowers students to experiment and explore data-driven insights independently.

Future work on LAMBDA could focus on several key areas. First, enhancing LAMBDA's ability to seamlessly integrate and leverage large models from various domains for statistical analysis could significantly improve its capacity to tackle complex data analysis tasks. Second, improving the user interface and increasing user satisfaction would make the system more accessible to non-experts. Third, incorporating real-time data processing capabilities could enable LAMBDA to handle streaming data, which is increasingly important in many applications. Finally, expanding the system's support for collaborative work among multiple users could further enhance its utility in both educational and professional settings. We plan to implement LAMBDA in our classroom teaching scenarios, continuously gather feedback from various groups, and use user satisfaction as a metric for evaluating LAMBDA.

In conclusion, LAMBDA represents a meaningful step forward in integrating human and artificial intelligence for data analysis. Its continued development and refinement have the potential to advance the fields of statistics and data science, making sophisticated analytical tools more accessible to users from diverse backgrounds. We have made our code available at https://github.com/AMA-CMFAI/LAMBDA.



# Acknowledgments

The authors are grateful to the Editor, Associate Editor, three anonymous reviewers, and the reproducibility reviewer for their valuable comments and suggestions, which significantly improved the quality of the paper.

## Funding

This work was funded by the Centre for the Mathematical Foundations of Generative AI and the research grants from The Hong Kong Polytechnic University (P0046811). The research of Ruijian Han was partially supported by The Hong Kong Polytechnic University (P0044617, P0045351, P0050935). The research of Houduo Qi was partially supported by the Hong Kong RGC grant (15309223) and The Hong Kong Polytechnic University (P0045347). The research of Defeng Sun and Yancheng Yuan was partially supported by the Research Center for Intelligent Operations Research at The Hong Kong Polytechnic University (P0051214). The research of Jian Huang was partially supported by The Hong Kong Polytechnic University (P0042888, P0045417, P0045931).

## Disclosure Statement

The authors report there are no competing interests to declare.

## References

Aeberhard, S. and Forina, M. (1991). The Wine dataset. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5PC7J.

Almeida, T. A., Hidalgo, J. M. G., and Yamakami, A. (2011). Contributions to the study of sms spam filtering: New collection and results. https://www.dt.fee.unicamp.br/~tiago/smsspamcollection/.

Anh, P. (2023). Three high-dimensional genomic datasets. https://www.kaggle.com/datasets/anhpknu/high-dimensional-data/data.

Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., et al. (2023). Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Bavli, I., Ho, A., Mahal, R., and McKeown, M. J. (2024). Ethical concerns around privacy and data security in ai health monitoring for parkinson's disease: Insights from patients, family members, and healthcare professionals. *AI & SOCIETY*, pages 1–11.

Bentink, S., Haibe-Kains, B., Risch, T., Fan, J. B., Hirsch, M. S., Holton, K., Rubio, R., April, C., Chen, J., Wang, J., Lu, Y., Wickham-Garcia, E., Liu, J., Culhane, A. C., Drapkin, R., Quackenbush, J., and Birrer, M. J. (2012). Angiogenic mrna and microrna gene expression signature predicts a novel subtype of serous ovarian cancer. *PloS One*, 7(2):e30269.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., et al. (2022). Improving language models by retrieving from trillions of tokens. *arXiv preprint arXiv:2112.04426*.

Brooks, T., Pope, D., and Marcolini, M. (2014). Airfoil Self-Noise. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5VW2C.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Cao, R., Lei, F., Wu, H., Chen, J., Fu, Y., Gao, H., Xiong, X., Zhang, H., Hu, W., Mao, Y., Xie, T., Xu, H., Zhang, D., Wang, S., Sun, R., Yin, P., Xiong, C., Ni, A., Liu, Q., Zhong, V., Chen, L., Yu, K., and Yu, T. (2024). Spider2-v: How far are multimodal agents from automating data science and engineering workflows? In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C., editors, *Advances in Neural Information Processing Systems*, volume 37, pages 107703–107744. Curran Associates, Inc.

Chapyter (2023). Chapyter. https://github.com/chapyter/chapyter.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., et al. (2022). Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Colaprico, A., Silva, T. C., Olsen, C., Garofano, L., Cava, C., Garolini, D., Sabedot, T. S., and et al. (2015). Tcgabiolinks: An r/bioconductor package for integrative analysis of tcga data. *Nucleic Acids Research*, 44(8):e71–71.

Dash, T., Chitlangia, S., Ahuja, A., and Srinivasan, A. (2022). A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Sci. Rep.*, 12(1):1040.

Dinh, A., Miertschin, S., Young, A., and Mohanty, S. D. (2023). National Health and Nutrition Health Survey 2013-2014 (NHANES) Age Prediction Subset. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5BS66.

Du, Z., Qian, Y., Liu, X., Ding, M., Qiu, J., Yang, Z., and Tang, J. (2022). Glm: General language model pretraining with autoregressive blank infilling. In *ACL*, pages 320–335.

FHS (1948). Framingham heart study dataset. https://www.framinghamheartstudy.org.

Fisher, R. A. (1988). Iris. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C56C76.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., et al. (2023). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N. V., Wiest, O., and Zhang, X. (2024). Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Hammer, S. M., Katzenstein, D. A., Hughes, M. D., Gundacker, H., Schooley, R. T., Haubrich, R. H., et al. (1996). A trial comparing nucleoside monotherapy with combination therapy in hiv-infected adults with cd4 cell counts from 200 to 500 per cubic millimeter. aids clinical trials group study 175 study team. *N. Engl. J. Med.*, 335(15):1081–1090.

Hong, S., Lin, Y., Liu, B., Wu, B., Li, D., Chen, J., et al. (2024). Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679*.

Hong, S., Zheng, X., Chen, J., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., et al. (2023). Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Huang, D., Bu, Q., Zhang, J. M., Luck, M., and Cui, H. (2023a). Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*.

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., et al. (2023b). A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:22311.05232*.

Interpreter, O. (2023). Open interpreter. https://www.openinterpreter.com.

Janosi, A., Steinbrunn, W., Pfisterer, M., and Detrano, R. (1988). Heart Disease. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C52P4X.

Kaggle SAD (2016). Student admission dataset. https://www.kaggle.com/datasets/mohansacharya/graduate-admissions.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., et al. (2023). Efficient memory management for large language model serving with pagedattention. In *SOSP*, page 611–626.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. In *NeurIPS*, pages 9459–9474.

Mialon, G., Dessì, R., Lomeli, M., Nalmpantis, C., Pasunuru, R., Raileanu, R., Rozière, B., Schick, T., Dwivedi-Yu, J., Celikyilmaz, A., et al. (2023). Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

Nash, W., Sellers, T., Talbot, S., Cawthorn, A., and Ford, W. (1995). Abalone. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C55C7W.

Oakes, B. J., Famelis, M., and Sahraoui, H. (2024). Building domain-specific machine learning workflows: A conceptual framework for the state of the practice. *ACM Trans. Softw. Eng. Methodol.*, 33(4):1–50.

OpenAI (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Park, S., Wang, A. Y., Kawas, B., Liao, Q. V., Piorkowski, D., and Danilevsky, M. (2021). Facilitating knowledge sharing from domain experts to data scientists for building nlp models. In *Proceedings of the 26th International Conference on IUI*, pages 585–596.

Pils, D., Hager, G., Tong, D., Aust, S., et al. (2012). Validating the impact of a molecular subtype in ovarian cancer on outcomes: a study of the ovcad consortium. *Cancer Science*, 103(7):1334–1341.

Piotrowski, T. J., Cavalcante, R. L., and Gabor, M. (2021). Fixed points of nonnegative neural networks. *arXiv preprint arXiv:2106.16239*.

Python Software Foundation (2023). Python: A programming language.

Qi, H. and Sun, D. (2006). A quadratically convergent newton method for computing the nearest correlation matrix. *SIAM J. Matrix Anal. Appl.*, 28(2):360–385.

Qiao, B., Li, L., Zhang, X., He, S., Kang, Y., Zhang, C., et al. (2023). Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*.

R Core Team (2023). R: A language and environment for statistical computing.

Rage, U. K., Pamalla, V., Toyoda, M., and Kitsuregawa, M. (2024). Pami: An open-source python library for pattern mining. *J. Mach. Learn. Res.*, 25(209):1–6.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP*, pages 3982–3992.

SAS Institute Inc. (2015). Sas/stat® 14.1 user's guide.

Sun, M., Han, R., Jiang, B., Qi, H., Sun, D., Yuan, Y., and Huang, J. (2024). A survey on large language model-based agents for statistics and data science. *arXiv preprint arXiv:2412.14222*.

Tfekci, P. and Kaya, H. (2014). Combined Cycle Power Plant. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5002N.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Tu, X., Zou, J., Su, W., and Zhang, L. (2024). What should data science education do with large language models? *Harvard Data Science Review*, 6(1).

Weihs, C. and Ickstadt, K. (2018). Data science: the impact of statistics. *Int. J. Data Sci. Anal.*, 6:189–194.

Weissgerber, T. L., Garovic, V. D., Milin-Lazovic, J. S., Winham, S. J., Obradovic, Z., Trzeciakowski, J. P., and Milic, N. M. (2016). Reinventing biostatistics education for basic scientists. *PLOS Biol.*, 14(4):e1002430.

Wolberg, W., Mangasarian, O., Street, N., and Street, W. (1995). Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5DW2B.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., et al. (2023). Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Yeh, I.-C. (2007). Concrete Compressive Strength. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5PK67.

Zhang, L., Zhang, Y., Ren, K., Li, D., and Yang, Y. (2023). Mlcopilot: Unleashing the power of large language models in solving machine learning tasks. *arXiv preprint arXiv:2304.14979*.

Zhang, T., Patil, S. G., Jain, N., Shen, S., Zaharia, M., Stoica, I., and Gonzalez, J. E. (2024). Raft: Adapting language model to domain specific rag. *arXiv preprint arXiv:2403.10131*.

Zheng, T., Zhang, G., Shen, T., Liu, X., Lin, B. Y., Fu, J., Chen, W., and Yue, X. (2024a). Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*.

Zheng, Y., Zhang, R., Zhang, J., Ye, Y., Luo, Z., Feng, Z., and Ma, Y. (2024b). Llamafactory: Unified efficient fine-tuning of 100+ language models. In *ACL*, pages 400–410.

Zhou, W., Jiang, Y. E., Li, L., Wu, J., Wang, T., Qiu, S., et al. (2023). Agents: An open-source framework for autonomous language agents. *arXiv preprint arXiv:2309.07870*.

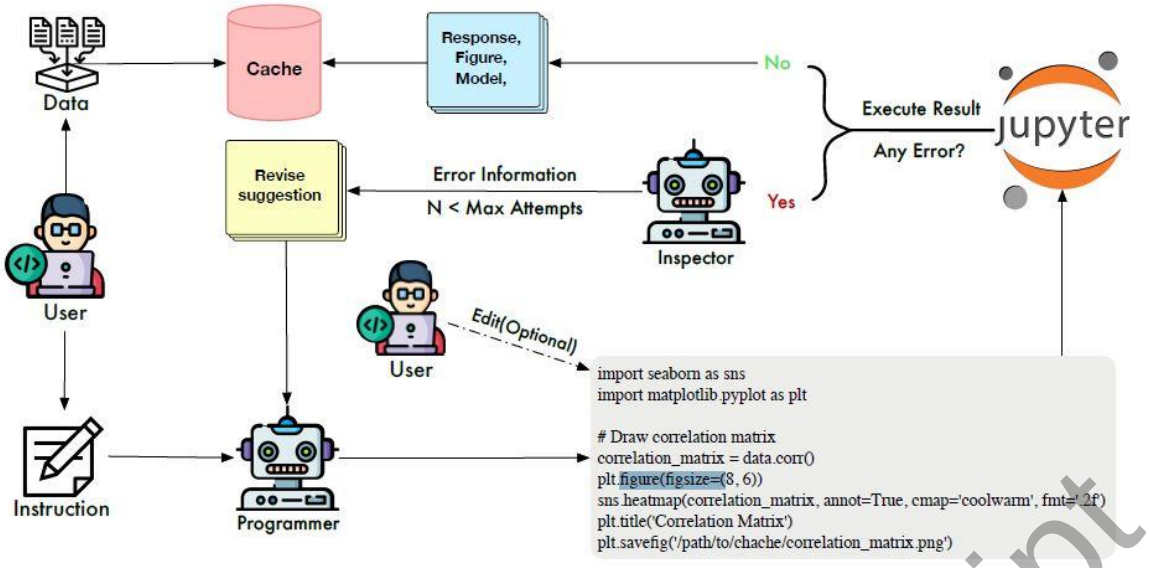


Figure 1: Overview of LAMBDA. LAMBDA features two core agents: the “programmer” for code generation and the “inspector” for error evaluation. The programmer writes and executes code based on user instructions, while the inspector suggests refinements if errors occur. This iterative process continues until the code is error-free or a maximum number of attempts is reached. A human intervention mechanism allows users to modify and run the code directly when needed.

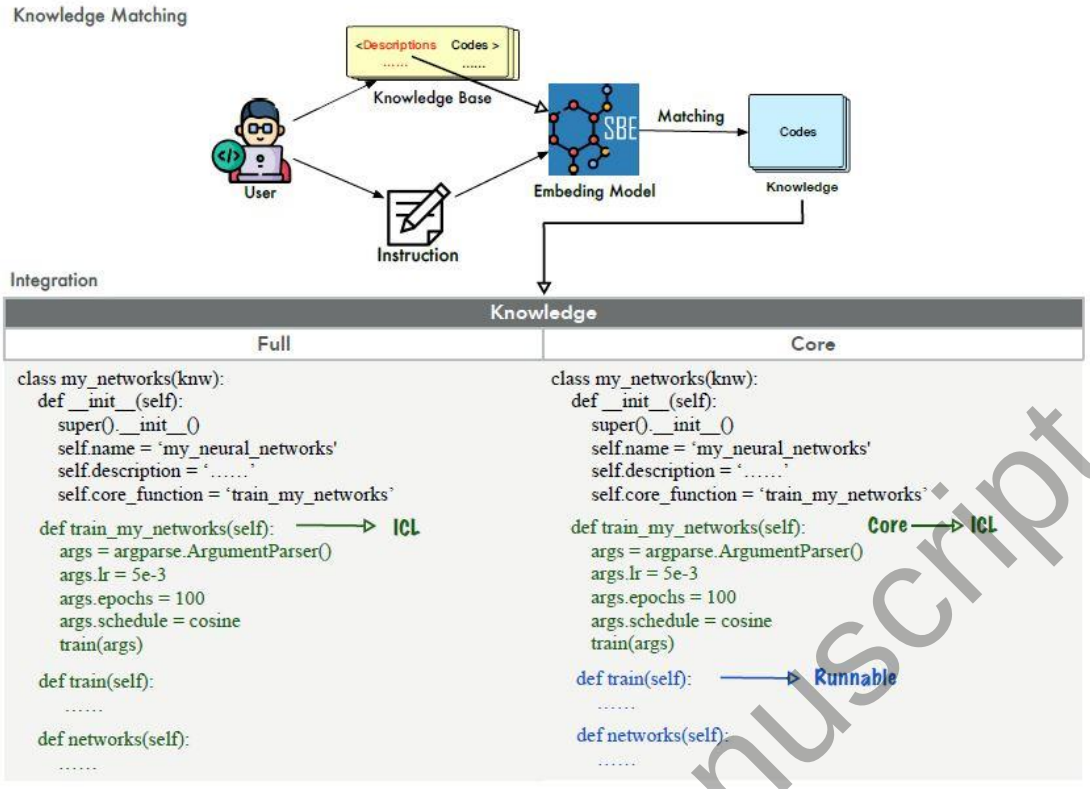


Figure 2: Knowledge Integration Mechanism in LAMBDA: Knowledge Matching selects codes from the knowledge base by comparing descriptions with the instruction. Two integration modes are available: 'Full' mode injects the entire knowledge code into the LLM via ICL, while 'Core' mode segments the code into essential usage for ICL and runnable code for back-end execution.

**System Prompt for Programmer**

You are a data analyst, your mission is to help humans do tasks related to data science and analysis. You are connecting to a computer. You should write Python code to complete the user's instructions. Since the computer will execute your code in Jupyter Notebook, you should directly use defined variables instead of rewriting repeated code. And your code should be started with markdown format like:
```python
**Write your code here.**
```
……You can work with data uploaded to your computer by users, the working path of the user is {**working_path**}. You must read or save files in this path. …… \nHere is an example:
{**example**}

Figure 3: Prompt example for the data analyst.

**System Prompt for Dataset**

Now, the user uploads the dataset in {**working_path**}, and here is the general information of the dataset:{
'**num_rows**': 150,
'**num_features**': 5,
'**features**': Index(['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width',\n 'Species'],\n dtype='object') ……
'**missing_val**': Sepal.Length 0\nSepal.Width 0\nPetal.Length 0\nPetal.Width 0\nSpecies 0\ndtype: int64,
'**describe**': Sepal.Length Sepal.Width Petal.Length Petal.Width
count 150.00 150.00 150.00 150.00
mean 5.84 3.06 3.76 1.20 ……}

Figure 4: Prompt example for the dataset.

**Prompt for Execution Result**

This is the execution result by the computer (If nothing is printed, it may be figures or files):
{**Executing_result**}.
You should use 1-3 sentences to explain or give suggestions for next steps:

Figure 5: Prompt example for the execution result.

**Prompt for Inspector**

You are an experienced and insightful inspector, and you need to identify the bugs in the given code based on the error messages and give modification suggestions.\n
- bug code:
{**bug_code**}\n
When executing the above code, errors occurred: {**error_message**}.
Please check the implementation of the function and provide a method for modification based on the error message. No need to provide the modified code.\n
Modification method:

Figure 6: Prompt example for inspector.

**Prompt for Programmer to Fix the Bug**

You should attempt to fix the bugs in the bellow code based on the provided error information and the method for modification. Please make sure to carefully check every potentially problematic area and make appropriate adjustments and corrections.
If the error is due to missing packages, you can install packages in the environment by "!pip install package_name".\n
- bug code:
{**bug_code**}\n
When executing the above code, errors occurred: {**error_message**}.
Please check and fix the code based on the modification method.\n
- modification method:
{**fix_method**}\n
The code you modified (should be wrapped in ```python```):

Figure 7: Prompt example for code correction.

**Prompt for Knowledge Integration**

**System Prompt for Retrieval**

You can retrieve codes from the knowledge base. The retrieved code will be formatted as:
Retrieval: The retriever finds the following pieces of code cloud address the problem:\n```python\n[retrieval_code]\n```
For example:
{example}

**Prompt for Retrieval**

Retrieval: The retriever finds the following pieces of code cloud address the problem. You should refer to this code and modify it as appropriate.
Retrival code:
{code}

Figure 8: Prompt example for knowledge integration.


Figure 9: An example of using LAMBDA for classification analysis with the Wine dataset.

Table 1: Datasets used in this study. The Genomic datasets include the following three datasets: TCGAmirna (Bentink et al., 2012), EMTAB386 (Colaprico et al., 2015), and GSE49997 (Pils et al., 2012).

| DataSets | Usage |
|---|---|
| AIDS Clinical Trials Group Study 175 (Hammer et al., 1996) | Classification |
| NHANES (Dinh et al., 2023). | Classification |
| Breast Cancer Wisconsin (Wolberg et al., 1995) | Classification |
| Wine (Aeberhard and Forina, 1991) | Classification |
| Concrete Compressive Strength (Yeh, 2007) | Regression |
| Combined Cycle Power Plant (Tfekci and Kaya, 2014) | Regression |
| Abalone (Nash et al., 1995) | Regression - Case Study |
| Airfoil Self-Noise (Brooks et al., 2014) | Regression |
| Iris (Fisher, 1988) | Classification - Case Study |
| Heart Disease (Janosi et al., 1988) | Regression - Case Study |
| Genomic Datasets (Anh, 2023) | High-Dimensional Data |
| Framingham Heart Study Dataset (FHS, 1948) | Missing Data |
| Student Admission Records (Kaggle SAD, 2016) | Missing Data |
| MINIST (LeCun et al., 1998) | Image Data |
| SMS Spam (Almeida et al., 2011) | Text Data |

Table 2: The experimental results obtained using LAMBDA and R are presented, with the R results indicated in parentheses. Classification problems were evaluated using accuracy, where higher values indicate better performance. Regression problems were assessed using mean squared error (MSE), where lower values are preferable. All results were derived from 5-fold cross-validation. The difference result bewteen LAMBDA and R is introduced by different data processing, hyper-paprameters and cross-validation.

| | **Model** | **Datasets** | | | |
|---|---|---|---|---|---|
| | | **AIDS** (%) | **NHANES** (%) | **Breast Cancer**(%) | **Wine**(%) |
| **Classification** | Logistic Regression | 86.54 (86.44) | 99.43 (99.96) | **98.07** (97.72) | **98.89** (98.86) |
| | SVM | 88.45 (88.59) | 98.82 (98.86) | 97.72 **(98.25)** | **98.89** (98.33) |
| | Neural Network | 88.82 (87.89) | 99.91 (99.91) | 97.82 (97.01) | 82.60 (98.87) |
| | Decision Tree | 87.70 (88.78) | **100 (100)** | 94.26 (93.32) | 92.14 (90.91) |
| | Random Forest | 89.29 (88.73) | **100 (100)** | 96.84 (95.96) | 98.33 (98.30) |
| | Bagging | 89.62 (88.82) | **100 (100)** | 96.49 (94.90) | 96.65 (96.60) |
| | Gradient Boost | 89.20 (88.83) | **100 (100)** | 96.84 (94.74) | 96.65 **(98.89)** |
| | XGBoost | **89.67** (**89.62**) | **100 (100)** | 97.54 (97.19) | 95.54 (98.87) |
| | AdaBoost | 88.92 (89.10) | **100 (100)** | 97.72 (97.55) | 93.89 (97.71) |
| | **Best Accuracy** | **89.67** (89.62) | **100 (100)** | 98.07 **(98.25)** | **98.89 (98.89)** |
| | | **Concrete** | **Power Plant** | **Abalone** | **Airfoil** |
| **Regression** | Linear Regression | 0.4596 (0.3924) | 0.0714 (0.0713) | 0.5086 (0.6867) | 0.5717 (0.6972) |
| | Lasso | 0.5609 (0.3918) | 0.0718 (0.0713) | 0.8042 (0.4739) | 0.5738 (0.4886) |
| | SVR | 0.4012 (0.4780) | 0.0534 (0.0489) | **0.4542 (0.4408)** | 0.3854 (0.3725) |
| | Neural Network | **0.2749** (0.3055) | 0.0612 (0.0567) | 0.4551 (0.7185) | 0.4292 (0.2604) |
| | Decision Tree | 0.5242 (0.5837) | 0.0551 (0.1175) | 0.5566 (0.5472) | 0.3823 (**0.2559**) |
| | Random Forest | 0.4211 (**0.2755**) | 0.0375 (**0.0363**) | 0.4749 (0.4460) | 0.2655 (0.3343) |
| | Gradient Boost | 0.3414 (0.3605) | **0.0315** (0.0538) | 0.4778 (0.5840) | **0.2528** (0.2888) |
| | XGBoost | 0.3221 (0.2991) | 0.0319 (0.0375) | 0.4778 (0.4441) | 0.2741 (0.2832) |
| | CatBoost | 0.2876 (0.4323) | 0.0325 (0.0568) | 0.4795 (0.4516) | 0.2529 (0.2638) |
| | **Best MSE** | **0.2749** (0.2755) | **0.0315** (0.0363) | (0.4542) **0.4408** | **0.2528** (0.2559) |

Table 3: Experiment datasets with their sizes and dimensions (rows, columns).

| Data | TCGAmirna | EMTAB386 | GSE49997 |
|---|---|---|---|
| (Size, Dimension) | (544, 802) | (129, 10360) | (194, 16051) |

Table 4: Performance on the high-dimensional datasets. The results are reported in terms of accuracy through 5-fold cross-validation.

| Model | TCGAmirna (%) | EMTAB386 (%) | GSE49997 (%) |
|---|---|---|---|
| Logistic Regression | 52.58 | 54.18 | 67.52 |
| Decision Tree | 54.42 | 57.45 | 63.45 |
| Random Forest | 55.16 | 61.20 | 67.54 |
| Bagging | **56.62** | 58.21 | **70.63** |
| Gradient Boosting | 54.78 | 55.08 | 70.62 |
| XGBoost | 55.15 | 58.15 | 70.62 |
| AdaBoost | 55.15 | 57.45 | 70.62 |
| Neural Network | 54.22 | **61.23** | 66.48 |
| Best | **56.62** | **61.23** | **70.63** |

Table 5: Performance on Framingham, StuRecord and Heart Disease datasets. The results are reported in terms of accuracy through 5-fold cross-validation.

| Model | Framingham (%) | StuRecord (%) | Heart Disease (%) |
|---|---|---|---|
| Logistic Regression | **85.35** | 50.36 | 59.41 |
| Neural Network | 84.95 | 57.28 | **60.40** |
| Decision Tree | 84.27 | 52.96 | 52.49 |
| Random Forest | 85.19 | 55.40 | 60.39 |
| Bagging | 85.02 | 58.65 | 60.06 |
| Gradient Boosting | 85.12 | 60.50 | 58.41 |
| XGBoost | 85.19 | **61.05** | 60.71 |
| AdaBoost | 84.98 | 56.63 | 59.42 |
| Best | **85.35** | **61.05** | **60.40** |

Table 6: Performance on the MNIST Dataset.

| Model | Accuracy (%) |
|---|---|
| CNN | 99.19 |
| Transformer | 97.23 |

Table 7: Performance of different backbones on the SPAM classification task.

| Model | Accuracy (%) |
|---|---|
| Multinomial Naive Bayes | 98.39 |
| BERT | 99.37 |

Table 8: Performance of the Knowledge Integration Mechanism. In the table, 'PM' refers to pattern mining, 'NCM' refers to the nearest correlation matrix, and 'FPNENN' stands for fixed points in non-negative neural networks. The values represent the performance scores, with failure reasons noted in brackets. Specifically, 1: code error and execution error; 2: exceeded runtime limit; 3: code error but successful execution; 4: right code but execution error due to other issues; 5: right code and successful execution.

| | PM | NCM | FPNENN |
|---|---|---|---|
| GPT-4-Advanced Data Analysis (OpenAI, 2023) | 0.80 (4) | 0 (1) | 0 (1) |
| ChatGLM-Data Analysis (Du et al., 2022) | 0 (2) | 0 (2) | 0 (2) |
| OpenInterpreter (Interpreter, 2023) | 0 (2) | 0 (2) | 0 (2) |
| OpenCodeInterpreter (Zheng et al., 2024a) | 1.00 (5) | 0 (1) | 0 (1) |
| Chapyter (Chapyter, 2023) | 0 (2) | 0 (2) | 0 (2) |
| DataInterpreter (Tools) (Hong et al., 2024) | **1.00 (5)** | **1.00 (5)** | **1.00 (5)** |
| TaskWeaver (Plugins) (Qiao et al., 2023) | **1.00 (5)** | **1.00 (5)** | **1.00 (5)** |
| **LAMBDA (Knowledge)** | **1.00 (5)** | **1.00 (5)** | **1.00 (5)** |

Table 9: The results of case study on Misalignment between Tools and Instructions. Both Plugins and Tools Integration directly use the tools and are not aware of the Misalignment between Tools and Instructions.

| **Methods** | **Misalignment Loss** | **Misalignment Network** |
|---|---|---|
| TaskWeaver (Plugins) | ✗ Directly using the plugin | ✗ Directly using the plugin |
| Data Interpreter (Tools) | ✗ Directly use the tool | ✗ Directly use the tool |
| LAMBDA (Knowledge) | ✓ Alignment | ✓ Alignment |