

LAMBDA: A Large Model Based Data Agent

Maojun Sun^a, Ruijian Han^b, Binyan Jiang^b,

Houduo Qi^{a,b}, Defeng Sun^a, Yancheng Yuan^{b*} and Jian Huang^{a,b*}

^aDepartment of Applied Mathematics, The Hong Kong Polytechnic University

^bDepartment of Data Science and Artificial Intelligence, The Hong Kong Polytechnic University

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 models. LAMBDA is designed to address data analysis challenges in complex data-driven applications through innovatively designed data agents that operate iteratively and generatively 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, enhanced by advanced models. Meanwhile, the *inspector* debugs the code when necessary. To ensure robustness and handle adverse scenarios, LAMBDA features a user interface that allows direct user intervention in the operational loop. Additionally, 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. Videos of several case studies are available at https://xxxlambda.github.io/lambda_webpage.

Keywords: code generation via natural language; data analysis; large models; multi-agent collaboration; software system.

*Corresponding authors.

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. Additionally, concerns regarding data privacy (Bavli et al., 2024) and security risks are inherent in the present configuration of GPT-4. In contrast, by utilizing the open-sourced LAMBDA, users can eliminate concerns regarding data privacy while benefiting from enhanced flexibility and convenience in integrating domain knowledge, installing packages, and utilizing diverse computational resources.

LAMBDA excels in performance across multiple datasets utilized in our system testing. Furthermore, it outperforms other agent systems in handling complex domain tasks during

our experiments. In summary, LAMBDA’s key features include: (1) a coding-free, natural language interface, (2) seamless integration of human intelligence and AI, (3) high reliability, and (4) automatic generation of analysis reports and code exporting.

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. Additionally, Section 5 illustrates case studies of LAMBDA in various scenarios, including data analysis, integration of human intelligence, and interactive education. The paper concludes with a summary in Section 6. Supplementary materials, including an alternative code generation approach, prompts, case studies, and experimental settings, are provided in the Appendix.

2 Background and related works

In recent years, the rapid progress in LLMs has brought boundless possibilities to the field of artificial intelligence and its applications in many fields, including statistics and data science. Notable examples of LLMs include GPT-3 (Brown et al., 2020), GPT-4 (OpenAI, 2023), PaLM (Chowdhery et al., 2022), LLaMA (Touvron et al., 2023) and Qwen (Bai et al., 2023). LLMs demonstrate an outstanding ability to understand, generate, and apply natural languages. 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, recommender systems, and scientific experiments (Guo et al., 2024; Wu et al., 2023; Zhou et al., 2023b).

2.1 LLMs as data analysis agents

Recently, significant efforts have been devoted to leveraging the power of LLMs to automate data science and analysis tasks (OpenAI, 2023). Chapter (chapyter, 2023) and Jupyter-AI (jupyterlab, 2023) integrate ChatGPT into Jupyter notebooks, enabling seamless user interaction. Some works have designed independent data science agent systems. For example, Data-Copilot (Zhang et al., 2023) and MatPlotAgent (Yang et al., 2024) have made significant

progress in data visualization. Data Interpreter (Hong et al., 2024) proposed a dynamic planning framework with hierarchical structures that demonstrate notable effectiveness in data science applications. Nonetheless, these works have not adequately addressed the high degree of user freedom required 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 Enhancing LLMs with function calling and code interpreter

The integration of external APIs or tools into LLMs, known as function calling, enables LLMs to utilize these tools to handle tasks through their functional capabilities (Chen et al., 2024; Kim et al., 2023). This process can be summarized as follows: First, LLMs classify the user’s instruction into functions based on function annotations. Then, the program executes the selected functions, and the LLM generates a final answer based on the execution results (Kim et al., 2023; Mialon et al., 2023). Qu et al. (2024) investigates the current paradigm of tool learning with LLMs and demonstrates its effectiveness in diverse applications such as programming, calculators, and weather inquiries. However, in the context of statistical and data science applications, the function-calling method may not perform well due to the following considerations:

- *Consideration A:* In applications, a large number of APIs are typically required. These APIs often have complex interrelationships and extensive annotations. Such lengthy API annotations can result in sequences that exceed the maximum processing capacity of current LLMs, leading to the truncation of context.
- *Consideration B:* The model’s ability to accurately select APIs diminishes as the number of available APIs increases. This decline is due to the increased complexity introduced by the growing number of APIs that LLMs need to evaluate. An incorrect choice of tools or models can directly lead to erroneous results and answers.

We will conduct experiments in Section 4 to evaluate the function-calling method and provide a detailed explanation for why we chose not to adopt this approach.

Equipping LLMs with code interpreters enables them to execute code and obtain execution results (Gao et al., 2023a; Mialon et al., 2023). Studies by Zeng et al. (2022); Du et al. (2022) and Zheng et al. (2024a) demonstrate the code interpreter capabilities in ChatGLM, making it accessible for tasks such as complex calculations and drawing figures. Zhou et al. (2023a) utilized the code interpreter to solve mathematical problems, achieving significant progress. However, the code required for statistical and data science problems is more complex and challenging than in the aforementioned domains. Additionally, as a general-purpose platform for statistical and data science applications, the system needs to handle diverse and heterogeneous tasks and data types. The accuracy of the code is crucial, as it directly impacts the reliability of the agent system. To address this issue, we propose a self-correction mechanism through multi-agent collaboration, which enhances reliability by enabling our agent to learn from its failures.

2.3 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). Qian et al. (2023a,b, 2024) proposed an agent system modeled after a software company, consisting of agents such as programmers, code reviewers, and test engineers, enabling the completion of the entire software development process rapidly. Li et al. (2024) introduced an agent hospital that simulates the entire process of treating illnesses, achieving state-of-the-art results in managing respiratory diseases. Similarly, LAMBDA simulates real-world data analysis workflows through the collaborative efforts of a data scientist agent and an inspector agent, thereby enhancing the system’s reliability.

2.4 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., 2023b), 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. We address these challenges through the development of a specially designed KV knowledge base with integration methods. It allows users to select different modes of retrieval and knowledge integration, according to the complexity, length of the knowledge context, and specific requirements of the task at hand. By knowledge integration, our agent system is adaptive for more domain tasks by human intelligence.

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. Additionally, an alternative function-calling-based agent system is illustrated in the Supplementary Materials. Through a comparative study,

we will discuss the limitations of the function-calling method and the advantages of the multi-agent collaboration system in Section 4.1.

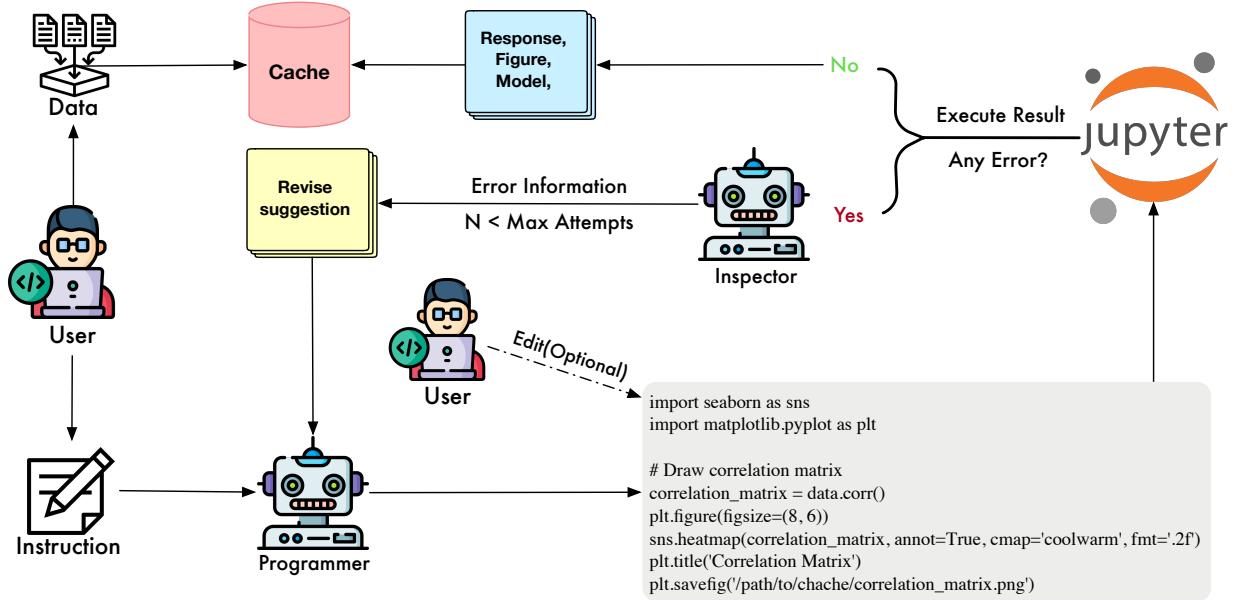


Figure 1: Overview of LAMBDA. LAMBDA features two core agents: the “programmer” for code generation and the “inspector” for error evaluation. The programmer writes and executes code based on user instructions, while the inspector suggests refinements if errors occur. This iterative process continues 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.

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. 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.

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 consisted 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.

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 data scientist
3:  $r = \begin{cases} r, & \text{success} \\ E, & \text{error} \end{cases} \leftarrow execute(C_n)$       ▷ 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:    $C_n \leftarrow A_n$ ,  $A_n \leftarrow Pr(C_{n-1}, S_n, E)$     ▷ Data Scientist modifies code
8:    $r \leftarrow execute(C_n)$                             ▷ Execute modified code
9: end while
10: if  $r = E$  then
11:    $r \leftarrow execute(C_h)$                           ▷ 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

```

A case of self-correcting mechanism is demonstrated in the Supplementary Materials. Our experiments result in Table 2 demonstrate that the agent system incorporating inspector significantly enhances the reliability.

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.

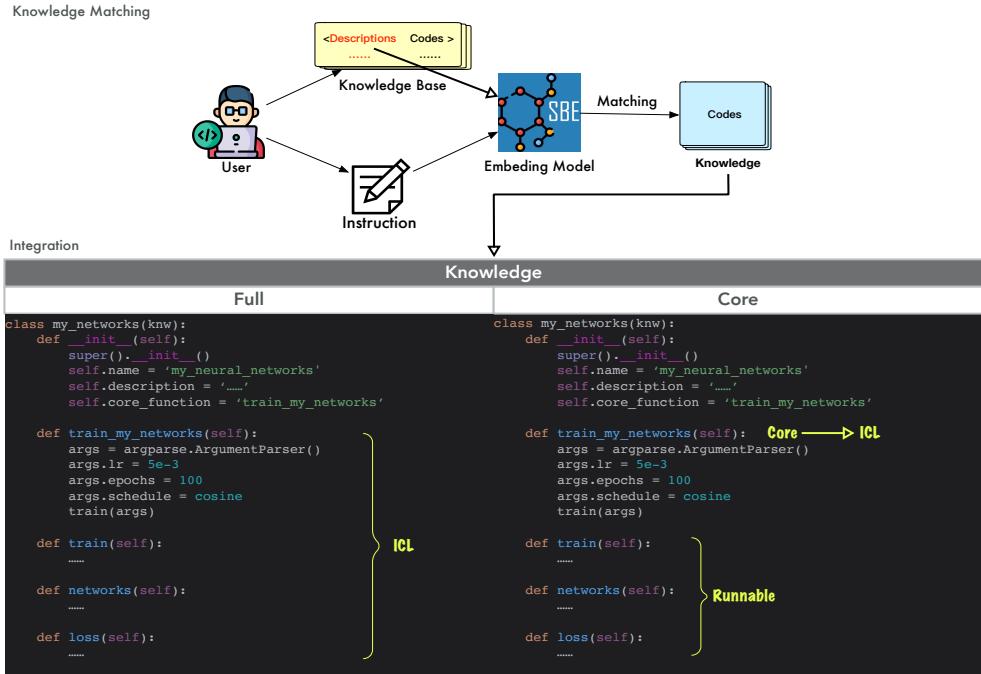


Figure 2: Knowledge Integration Mechanism in LAMBDA: Knowledge Matching selects codes from the knowledge base by comparing descriptions with the instruction. Two integration modes are available: ‘Full’ mode injects the entire knowledge code into the LLM via ICL, while ‘Core’ mode segments the code into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

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 $\mathcal{K} = \{(d_i, c_i) \mid i = 1, 2, \dots, n\}$, 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 the knowledge with a similarity score greater than a threshold θ and the highest match as the knowledge.

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

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

We let the matching threshold $\theta = 0.5$. The matched knowledge k with the highest S_i is selected while it satisfies $S_i > \theta$, computed as

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

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 $D = \{(q_1, k_1, a_1), (q_2, k_2, a_2), \dots, (q_n, k_n, a_n)\}$, and the LLM \mathcal{M} , the model estimates the probability $\mathcal{P}(a|q, k, D)$ and outputs the answer \hat{A} that maximizes this probability. The final response \hat{A} is generated by the model \mathcal{M} as $\hat{A} \leftarrow \mathcal{M}(q, D)$.

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 4 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 employs IPython as the system’s kernel because it is a common practice in data analysis to process data sequentially, where current operations build upon previous ones. For instance, standardization is typically performed first, followed by one-hot encoding, with the latter relying on the data that has already been standardized. This sequential relationship is effectively managed in IPython. The implementation details of the kernel can be found in the Supplementary Materials. Additionally, LAMBDA can generate analysis reports based on the dialogue history. These reports typically include data processing steps, data visualizations, model descriptions, and evaluation results. LAMBDA provides users with a selection of report templates, and the agent generates reports in the desired format through in-context learning (ICL). This feature enables users to concentrate on higher-value tasks. A sample report can be found in Figure 11 and the Supplementary Materials. Furthermore, LAMBDA allows users to download their experimental code as an IPython notebook, accommodating varying user preferences.

3.6 User interface

LAMBDA offers an accessible user experience through its intuitive interface, which resembles that of ChatGPT. Users can begin by uploading their datasets and describing their desired tasks in natural language. The system supports multiple languages, facilitated by LLMs

such as Qwen-2, which recognizes 27 languages. It is recommended to prompt LAMBDA step-by-step, emulating the approach typically used by data analysts, rather than issuing a single comprehensive task description. This method allows users to maintain control over the process, embodying the “human-in-the-loop” concept. Once LAMBDA generates results, including code, figures, and models, users can easily copy and save these resources with a single click. Even individuals without expertise in statistics or data science can leverage LAMBDA to train advanced models by simply requesting, for example, “Recommend some models for me.” LAMBDA can suggest advanced models like XGBoost and AdaBoost, which may be unfamiliar to average users. Advanced users can further customize LAMBDA’s knowledge using an interface template. Additionally, users can export text reports and code for further study or experimentation. A usage example of how LAMBDA interacts with users is presented in Figure 11. Overall, LAMBDA’s user interface is designed to be accessible to a broad range of users, regardless of their background in statistics or data science.

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 some 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.

An example prompt for the data analyst at the start of the analysis session is given in Figure 3.

System Prompt for Programmer

You are a data analyst, your mission is to help humans do tasks related to data science and analytics. You are connecting to a computer, but there is no internet connection. 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.

Here is an example:

{example}

Figure 3: Prompt example for the data analyst.

A system prompt for the dataset can provide essential information to the programmer agent. (Figure 4.)

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', 'Species'], dtype='object'), 'col_type': Sepal.Length
   float64\nSepal.Width  float64\nPetal.Length  float64\nPetal.Width  float64\nSpecies  object\ndtype: 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\ncount  150.000000  150.000000  150.000000
  150.000000\nmean  5.843333  3.057333  3.758000  1.199333\nstd  0.828066  0.435866  1.765298  0.762238\nmin
 4.300000  2.000000  1.000000  0.100000\n25%  5.100000  2.800000  1.600000  0.300000\n50%  5.800000
 3.000000  4.350000  1.300000\n75%  6.400000  3.300000  5.100000  1.800000\nmax  7.900000  4.400000
 6.900000  2.500000}"}
```

Figure 4: Prompt example for the dataset.

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.

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.

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).

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.

- bug code:
{bug_code}

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.

Modification method:

Figure 6: Prompt example for inspector.

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

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”.

- bug code:
{bug_code}

When executing the above code, errors occurred: {error_message}.
Please check and fix the code based on the modification method.

- modification method:
{fix_method}

The code you modified (should be wrapped in `python`):

Figure 7: Prompt example for code correction.

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

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 if there is a test case, you can make changes directly to the test case instead of re-writing all functions:

Retrieval code:

{code}

Figure 8: Prompt example for knowledge integration.

4 Experiments

We first evaluate the considerations of function-calling method described in Section 2.2. We then conduct an ablation study on LAMBDA to demonstrate the impact and performance of each agent. Furthermore, to assess the effectiveness of LAMBDA, we evaluate its performance on several datasets and compare it with artificiality. Lastly, we compare LAMBDA with other advanced agents, such as GPT-4, on domain-specific tasks to highlight the effectiveness of the Knowledge Integration Mechanism.

4.1 Challenges of the function-calling method

We estimate the maximum number of APIs that some open-source LLMs can handle in data analysis by using the average length of the pre-defined APIs. Figure 9 illustrates the results. Qwen1.5 and Mistral-v0.1 (Jiang et al., 2023) were specifically designed to handle lengthy sequences, capable of managing 400 and 372 APIs, respectively. However, general-purpose LLMs such as LLaMA2, LLaMA3, Mistral-V0.2, Qwen1, ChatGLM2, and ChatGLM3 can process fewer than 100 APIs. This limitation poses a challenge for applications requiring a larger number of APIs, such as data analysis tasks.

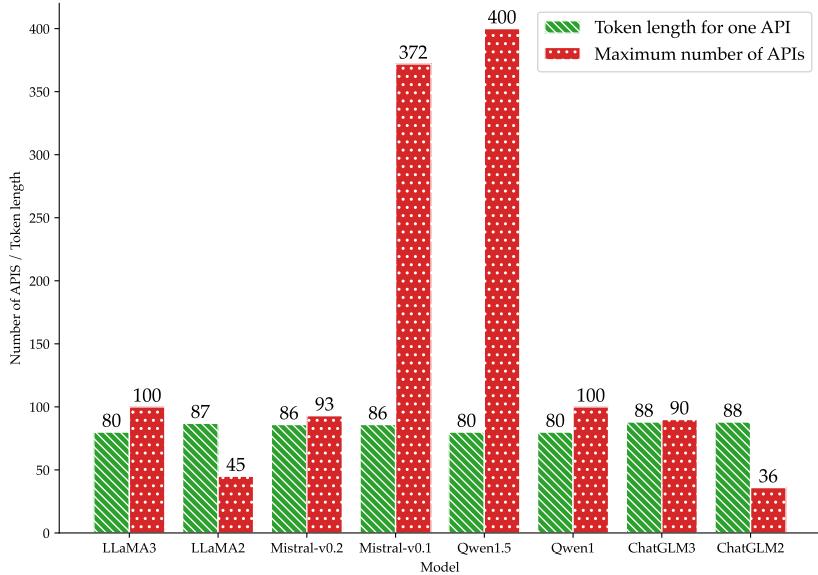


Figure 9: Average token length for one API and maximum number of APIs each LLM can process.

To investigate the impact of the number of APIs on the accuracy of LLMs in selecting the correct APIs, we constructed a dataset comprising 100 commonly used APIs in data analysis. Using few-shot learning, we generated 880 testing instructions aligned with these APIs using Qwen1.5-110B. We then segmented both the APIs and testing instructions into intervals of 10 functions for analysis. The details of the evaluation dataset are shown in Table 1, and the results are presented in Figure 10.

Table 1: Number of APIs and corresponding instructions in the evaluation dataset.

	APIs									
	10	20	30	40	50	60	70	80	90	100
Instructions	74	163	268	352	446	525	614	684	806	880

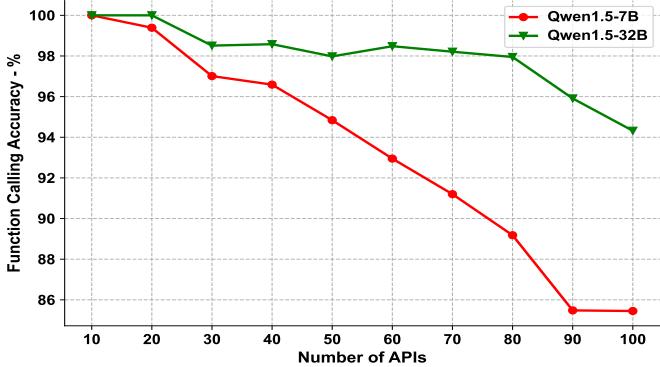


Figure 10: The accuracy of API chosen by model Qwen1.5. Qwen1.5 is used for the experiments due to its capability of processing the maximum number of APIs in our experiments.

The results of API selection indicate a notable decline in the model’s ability to accurately select APIs as the number of APIs increases. Specifically, the accuracy decreased from 100% to 94.32% and 85.45% for Qwen1.5-32b and Qwen1.5-7b respectively. In the data science scenario, the number of APIs can be extensive due to it encompassing various processing methods and combinations.

In summary, the function-calling method exhibits several significant drawbacks. Firstly, the labor-intensive process of defining numerous APIs proves inefficient. Secondly, the static nature of APIs hinders adaptability to diverse and evolving user demands. Thirdly, extensive API annotations can occupy a substantial portion of the input sequence, potentially leading to truncation risks. Lastly, as the number of APIs increases, the model’s accuracy in correct selection decreases, thereby introducing potential inaccuracies in results.

4.2 Evaluation of the inspector agent

To assess the importance of the inspector agent in LAMBDA, we designed an ablation study using the heart disease dataset (Janosi et al., 1988). This dataset, which contains missing values, naturally presents challenges. We utilized Qwen1.5-110B to generate instructions for related tasks. After filtering, there were 454 instructions in the experiment. We evaluated

the execution pass rate with only a single agent (programmer) and two agents (programmer and inspector), respectively. The results are summarized in Table 2.

Table 2: Experiment on single agent versus multiple agents. The percentages in brackets are the improvement rate over the single agent. Both the programmer and inspector agent are implemented by Qwen1.5-32b in this experiment.

Agents	Passing Rate %
programmer agent only	68.06
programmer + inspector	95.37 (40.13%)

The result shows a significant gap in the passing rate between using a programmer agent alone and incorporating an inspector. The programmer agent achieved a passing rate of 68.06%, while the integration of the inspector increased the passing rate to 95.37%, marking a substantial improvement of 40.13% over the single-agent setup. This experiment verified the crucial role of collaborative agents in enhancing the reliability and robustness of LAMBDA. By leveraging complementary strengths in error suggestion and correction, the multi-agent collaboration approach not only improves the code passing rate but also reduces the frequency of human intervention in executing data analysis tasks.

4.3 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 applied LAMBDA to several standard datasets for classification and regression analysis. For comparison, we also conducted the analysis using R with the same models. In classification problems, the classification accuracy was used for the test data, which is defined as the ratio of the number of correctly classified instances to the total number of instances. For regression problems, Mean Squared Error (MSE) was used, which is defined as the average of the

squared differences between the predicted values and the actual values in the test data. It is given by the formula: $MSE = (1/n) \sum_{i=1}^n (y_i - \hat{y}_i)^2$, where n is number of data points, y_i is the observed value, \hat{y}_i is the predicted value. We used 5-fold cross validation in evaluation in all the cases. The results are summarized in Table 3, with the corresponding results from R provided in parentheses.

We use the following datasets in the experiments.

- *AIDS Clinical Trials Group Study 175* provides healthcare statistics and categorical data on AIDS patients. It includes 2139 instances with 23 features, primarily used for predicting patient mortality within a specified time frame (Hammer et al., 1996).
- *National Health and Nutrition Health Survey 2013-2014 Age Prediction Subset (NHANES)* is derived from the CDC's comprehensive health and nutrition survey. This subset, with 6287 instances and 7 features, focuses on predicting respondents' age groups using physiological, lifestyle, and biochemical data (Dinh et al., 2023).
- *Breast Cancer Wisconsin (Diagnostic)* comprises 569 instances and 30 features. It classifies patients into categories of malignant or benign (Wolberg et al., 1995).
- *Wine* contains the results of a chemical analysis of wines from a specific region in Italy, derived from three different cultivars. It includes 178 instances and 13 features, focusing on the quantities of various constituents found in the wines (Aeberhard and Forina, 1991).
- *Concrete Compressive Strength* contains 1030 instances with 8 features, examining the highly nonlinear relationship between concrete compressive strength, age, and ingredients (Yeh, 2007).
- *Combined Cycle Power Plant* features 9568 data points collected over six years, with 4 features, to analyze the performance of a power plant under full load conditions (Tfekci and Kaya, 2014).
- *Abalone* contains 4177 instances with 8 features for building a model predicting the age of abalone from physical measurements (Nash et al., 1995).
- *Airfoil Self-Noise* includes 1503 instances with 5 features derived from aerodynamic and acoustic tests of airfoil blade sections in an anechoic wind tunnel. This dataset can be used to train a model for predicting scaled sound pressure, (Brooks et al., 2014).

Table 3: 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.

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)
Regression	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)

The results presented in Table 3 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 diverse data science 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 across various domains in handling data analysis tasks.

4.4 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 latest research 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 S that is calculated as follows:

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

Other agents are also augmented by one-shot learning to maintain consistency in the experiment. All agent is implemented by 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 have set a maximum runtime limit of 5 minutes to prevent some agents from getting stuck in infinite self-modification loops.

- *Pattern Mining* Piotrowski et al. (2021) present PAMI (PAttern MIning), a cross-platform open-source Python library that provides several algorithms to discover

different types of patterns hidden in various types of databases across multiple computing architectures. The task is easy by giving an example in context.

- *Nearest Correlation Matrix* Qi and Sun (2006) propose a Newton-type method specifically designed for the nearest correlation matrix problem. Leveraging recent developments related to strongly semi-smooth matrix-valued functions, they prove the quadratic convergence of their proposed Newton method. 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. The task is to train such a neural network with a given network architecture.

Table 4: Performance and Comparative Study of the Knowledge Integration Mechanism. 'PM' refers to pattern mining, 'NCM' refers to the nearest correlation matrix, and 'FPNENN' stands for fixed points in non-negative neural networks.

	PM	NCM	FPNENN	Average
GPT-4-Advanced Data Analysis (OpenAI, 2023)	0.80	0	0	0.27
ChatGLM-Data Analysis (Du et al., 2022)	0	0	0	0
OpenInterpreter (Interpreter, 2023)	0	0	0	0
OpenCodeInterpreter (Zheng et al., 2024a)	1.00	0	0	0.33
DataInterpreter (Hong et al., 2024)	1.00	0	1.00	0.67
JupyterAI (jupyterlab, 2023)	0	0	0	0
Chapyter (chapyter, 2023)	0	0	0	0
TaskWeaver (Qiao et al., 2023)	1.00	0	0	0.33
LAMBDA	1.00	1.00	1.00	1.00

The results presented in Table 4 underscore the superior effectiveness of the knowledge integration mechanism. While other methods are executed one-shot prompting, many fail to perform adequately due to insufficient knowledge or an inability to generate correct code and handle errors. Even the advanced GPT-4 encounters difficulties with package installation

because of its sandbox mechanism, highlighting the limitations of closed-source agents. In contrast, LAMBDA demonstrates a clear advantage over other agents, successfully completing all tasks related to pattern mining, nearest correlation matrices, and fixed points in non-negative neural networks. This experiment further illustrates that Knowledge Integration is a valuable method for incorporating human intelligence and bridging the gap between domain experts and data scientists.

5 Case studies

We present an example of using LAMBDA for building a classification model in Figure 11. 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 available at <https://www.youtube.com/watch?v=fGvXWWeUH8A>.
- *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 in performing this task, thereby underscoring the value of LAMBDA through comparison. A demonstration of this process is available at <https://www.youtube.com/watch?v=sfU2ZzvNke0>.
- *Interactive Education* We consider an educational scenario in which the teacher uses LAMBDA to design the curriculum, 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 of this process is available at <https://www.youtube.com/watch?v=8q-EFoak9ZI&t=20s>.

This is the Wine dataset. I want to do a classification task, can you recommend some models for me?

For your classification task, here are some recommended models, along with brief explanations of their strengths:

1. Logistic Regression
A simple and interpretable linear model, effective for binary classification tasks, and can be extended to multiclass problems using softmax or one-vs-rest approaches.
2. Support Vector Machine (SVM)
Very effective in high-dimensional spaces and with non-linear data using the kernel trick. It's useful for classification with a clear margin of separation.
3. Neural Network (MLPClassifier)
A powerful model capable of capturing complex patterns, especially in large datasets. Multilayer perceptrons can be used for classification problems by learning non-linear relationships.
4. XGBoost
A highly efficient and scalable gradient-boosting model that uses advanced regularization techniques to prevent overfitting. It's often the top performer in structured data classification tasks.
5. AdaBoost
Combines weak classifiers (like decision stumps) by giving more weight to misclassified instances, allowing it to focus on difficult cases in the data.

Let me know if you need specific guidance on setting up any of these models!

My target column is class. First, draw a correlation matrix for the data.

```
import pandas as pd
# Load the dataset
data = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.csv')
# Compute the correlation matrix
correlation_matrix = data.corr()
# Visualize the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')

Here is the correlation matrix for the wine dataset, with the target column being 'class'. This visualization highlights the relationships between the features and how they correlate with each other and the target.
```

Standardize the data. Then, encode the column class ordinal.

```
import pandas as pd
# Load the dataset
data = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.csv')
# Standardize the data
data[['Alcohol', 'Malic acid', 'Ash', 'Alcalinity_of_ash', 'Magnesium', 'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols', 'Proanthocyanins', 'Color_intensity', 'Hue', 'OD315_of_diluted_wines', 'Proline']] = scaler.fit_transform(data[['Alcohol', 'Malic acid', 'Ash', 'Alcalinity_of_ash', 'Magnesium', 'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols', 'Proanthocyanins', 'Color_intensity', 'Hue', 'OD315_of_diluted_wines', 'Proline']])

Execution result:
Here is the rewritten table in Markdown format:
| | Alcohol | Malic acid | Ash | Alcalinity_of_ash | Magnesium | Total phenols | Flavanoids | Nonflavanoid_phenols | Proanthocyanins | Color_intensity | Hue | OD315_of_diluted_wines | Proline | class |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| 0 | 1.516813 | 0.562550 | 0.230254 | 1.169569 | 1.913905 | 0.808997 | 1.034819 | 0.659564 | 1.224884 | 0.251717 | 0.362177 | 0.847920 | 1.031309 | 0.0 |
| 1 | 1.516813 | 0.562550 | 0.230254 | 1.169569 | 1.913905 | 0.808997 | 1.034819 | 0.659564 | 1.224884 | 0.251717 | 0.362177 | 0.847920 | 1.031309 | 0.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 149 | 1.246290 | -0.499418 | -0.827998 | -2.696847 | 0.016184 | 0.568848 | 0.33629 | -0.2620719 | 0.544711 | 0.263321 | 0.406051 | 1.131349 | 0.965242 | 0.0 |

The standardized data appears to have a more normal distribution, which is suitable for many machine learning algorithms. The ordinal encoding for the class column has converted the categorical data into a numerical format, which can be used as input for many machine learning models. Next, you can split the data into training and testing sets, and then train a classification model using the standardized and encoded data.
```

Based on the processed data last step, perform cross-validation on the LogisticRegression model. Using accuracy to evaluate it.

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# Create a LogisticRegression object
logreg = LogisticRegression()
# Perform cross-validation
scores = cross_val_score(logreg, X, y, scoring='accuracy')

Execution result:
Score: 0.9722222222222222 1, 1, 1.
Average Score: 0.9888888888888889

The 5-fold cross-validation results show a high accuracy of 0.9889, indicating that the Logistic Regression model is performing well on the dataset. This suggests that the model is able to accurately classify the wine samples based on their characteristics. Next, you could consider tuning the hyperparameters of the Logistic Regression model to further improve its performance.
test_logreg_regression_code.py
```

Train and select parameters for SVM. Print the best parameters and score based on cross-validation.

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
# Create an SVC object
param_grid = {'C': [1, 10], 'gamma': [0.1, 1], 'kernel': ['rbf']}
# Perform grid search
grid_search = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X, y)

Execution result:
Best Parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
Best Score: 0.9888888888888889

The grid search results show that the best parameters for the SVM model are C=10, gamma='scale', and kernel='rbf', which achieved an accuracy of 0.9889. This suggests that the RBF kernel with a high regularization strength (C=10) and a scale-based gamma value is effective for this dataset. Next, you could consider using the best SVM model to make predictions on new, unseen data.
test_svm_model.py
```

Train and select parameters for XGBoost. Print the best parameters and score based on cross-validation.

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
# Create an XGBClassifier object
param_grid = {'learning_rate': [0.1, 0.5, 1], 'max_depth': [3, 5, 7], 'n_estimators': [10, 50, 100]}
# Perform grid search
grid_search = GridSearchCV(XGBClassifier(), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X, y)

Execution result:
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}
Best Score: 0.95542539625396254

The grid search results show that the best parameters for the XGBClassifier are learning_rate=0.1, max_depth=3, and n_estimators=100, which achieved an accuracy of 0.9554. This suggests that the XGBoost classifier with a learning rate of 0.1, a maximum depth of 3, and 100 estimators is effective for this dataset. Next, you could consider using the best XGBClassifier to make predictions on new, unseen data.
test_xgb_model.py
```

Report

Classification Task Using Wine Dataset with Machine Learning Models

1. Abstract:
This report details the process of building and evaluating machine learning models for a classification task on the Wine dataset. The dataset was chosen for its simplicity and the ability to demonstrate various machine learning concepts. The goal is to predict the wine class (Red or White) based on chemical properties. The report includes an overview of the dataset, a detailed description of the machine learning models used, and a comparison of their performance. It also discusses various evaluation metrics and provides recommendations for future work.

2. Introduction:
The task is to perform a classification on the Wine dataset, a well-known dataset that contains attributes related to different types of wine. The goal is to predict the type of wine based on its chemical properties. Machine learning models can help us to understand the data better and make predictions. In this task, we will learn how to build a machine learning model to predict the wine class. We will use various machine learning models and compare their performance. We will also discuss various evaluation metrics and provide recommendations for future work.

3. Methodology:
The methodology used in this report follows a standard machine learning workflow. We start by loading the dataset and performing initial exploratory data analysis (EDA). Then, we preprocess the data by handling missing values and encoding categorical variables. We then split the data into training and testing sets. Finally, we train several machine learning models (Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbors, SVM, and XGBoost) and evaluate their performance using various metrics (Accuracy, Precision, Recall, F1-score, etc.).

4. Results:
The results section presents the performance of the different machine learning models. The table below summarizes the accuracy of each model on the test set.

Model	Best Parameters	Accuracy
Logistic Regression	(C: 10, gamma: 'scale', kernel: 'rbf')	0.9889
Decision Tree	(criterion: 'entropy', max_depth: 3, min_samples_leaf: 1)	0.9813
Random Forest	(criterion: 'entropy', max_depth: 3, max_features: 'sqrt', n_estimators: 100)	0.9805
K-Nearest Neighbors	(n_neighbors: 3)	0.9805
SVM	(C: 10, gamma: 'scale', kernel: 'rbf')	0.9889
XGBoost	(learning_rate: 0.1, max_depth: 3, n_estimators: 100)	0.9554

5. Conclusion:
The report concludes with a summary of the findings and recommendations for future work. The main findings are that all models performed well, with the XGBoost model achieving the highest accuracy. Future work could involve exploring more complex models or feature engineering to further improve performance.

Classification Task Using Wine Dataset with Machine Learning Models

1. Abstract:
This report details the process of building and evaluating machine learning models for a classification task on the Wine dataset. The dataset was chosen for its simplicity and the ability to demonstrate various machine learning concepts. The goal is to predict the wine class (Red or White) based on chemical properties. The report includes an overview of the dataset, a detailed description of the machine learning models used, and a comparison of their performance. It also discusses various evaluation metrics and provides recommendations for future work.

2. Introduction:
The task is to perform a classification on the Wine dataset, a well-known dataset that contains attributes related to different types of wine. The goal is to predict the type of wine based on its chemical properties. Machine learning models can help us to understand the data better and make predictions. In this task, we will learn how to build a machine learning model to predict the wine class. We will use various machine learning models and compare their performance. We will also discuss various evaluation metrics and provide recommendations for future work.

3. Methodology:
The methodology used in this report follows a standard machine learning workflow. We start by loading the dataset and performing initial exploratory data analysis (EDA). Then, we preprocess the data by handling missing values and encoding categorical variables. We then split the data into training and testing sets. Finally, we train several machine learning models (Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbors, SVM, and XGBoost) and evaluate their performance using various metrics (Accuracy, Precision, Recall, F1-score, etc.).

4. Results:
The results section presents the performance of the different machine learning models. The table below summarizes the accuracy of each model on the test set.

Model	Best Parameters	Accuracy
Logistic Regression	(C: 10, gamma: 'scale', kernel: 'rbf')	0.9889
Decision Tree	(criterion: 'entropy', max_depth: 3, min_samples_leaf: 1)	0.9813
Random Forest	(criterion: 'entropy', max_depth: 3, max_features: 'sqrt', n_estimators: 100)	0.9805
K-Nearest Neighbors	(n_neighbors: 3)	0.9805
SVM	(C: 10, gamma: 'scale', kernel: 'rbf')	0.9889
XGBoost	(learning_rate: 0.1, max_depth: 3, n_estimators: 100)	0.9554

5. Conclusion:
The report concludes with a summary of the findings and recommendations for future work. The main findings are that all models performed well, with the XGBoost model achieving the highest accuracy. Future work could involve exploring more complex models or feature engineering to further improve performance.

Figure 11: An example of using LAMBDA for classification analysis with the Wine dataset.

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 preliminarily 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. Additionally, the system’s open-source nature encourages collaboration and continuous improvement from the global research community. As LAMBDA evolves, it has the potential to become a valuable tool for statisticians, data scientists, educators, and domain experts, contributing to advancements in knowledge and application in statistics and data science.

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 user experience 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.

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.

References

- Aeberhard, S. and Forina, M. (1991). Wine. UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C5PC7J>.
- Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., Hui, B., Ji, L., Li, M., Lin, J., Lin, R., Liu, D., Liu, G., Lu, C., Lu, K., Ma, J., Men, R., Ren, X., Ren, X., Tan, C., Tan, S., Tu, J., Wang, P., Wang, S., Wang, W., Wu, S., Xu, B., Xu, J., Yang, A., Yang, H., Yang, J., Yang, S., Yao, Y., Yu, B., Yuan, H., Yuan, Z., Zhang, J., Zhang, X., Zhang, Y., Zhang, Z., Zhou, C., Zhou, J., Zhou, X., and Zhu, T. (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.
- Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G., Lespiau, J.-B., Damoc, B., Clark, A., de Las Casas, D., Guy, A., Menick, J., Ring, R., Hennigan, T., Huang, S., Maggiore, L., Jones, C., Cassirer, A., Brock, A., Paganini, M., Irving, G., Vinyals, O., Osindero, S., Simonyan, K., Rae, J. W., Elsen, E., and Sifre, L. (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., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.
- chapyter (2023). Chapyter. <https://github.com/chapyter/chapyter>.

- Chen, W., Li, Z., and Ma, M. (2024). Octopus: On-device language model for function calling of software apis. *arXiv preprint arXiv:2404.01549*.
- Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2022). Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.
- Dash, T., Chitlangia, S., Ahuja, A., and Srinivasan, A. (2022). A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Scientific Reports*, 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 *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.
- Fisher, R. A. (1988). Iris. UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C56C76>.
- Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. (2023a). Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.
- Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., and Wang, H. (2023b). 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., Henry, W. K., Lederman, M. M., Phair, J. P., Niu, M., Hirsch, M. S., and Merigan, T. C. (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. *The New England journal of medicine*, 335 15:1081–90.

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

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., Chen, Q., Peng, W., Feng, X., Qin, B., and Liu, T. (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>.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. (2023). Mistral 7b. *arXiv preprint arXiv:2310.06825*.

jupyterlab (2023). Jupyter-ai. <https://github.com/jupyterlab/jupyter-ai>.

Kim, S., Moon, S., Tabrizi, R., Lee, N., Mahoney, M. W., Keutzer, K., and Gholami, A. (2023). An llm compiler for parallel function calling. *arXiv preprint arXiv:2312.04511*.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. (2023). Efficient memory management for large language model serving with

- pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP ’23, page 611–626, New York, NY, USA. Association for Computing Machinery.
- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.
- Li, J., Wang, S., Zhang, M., Li, W., Lai, Y., Kang, X., Ma, W., and Liu, Y. (2024). Agent hospital: A simulacrum of hospital with evolvable medical agents. *arXiv preprint arXiv:2405.02957*.
- 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 Transactions on Software Engineering and Methodology*, 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 Intelligent User Interfaces*, pages 585–596.
- 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 Journal on Matrix Analysis and Applications*, 28(2):360–385.
- Qian, C., Cong, X., Liu, W., Yang, C., Chen, W., Su, Y., Dang, Y., Li, J., Xu, J., Li, D., Liu, Z., and Sun, M. (2023a). Communicative agents for software development. *arXiv preprint arXiv:2307.07924*.
- Qian, C., Dang, Y., Li, J., Liu, W., Chen, W., Yang, C., Liu, Z., and Sun, M. (2023b). Experiential co-learning of software-developing agents. *arXiv preprint arXiv:2312.17025*.
- Qian, C., Li, J., Dang, Y., Liu, W., Wang, Y., Xie, Z., Chen, W., Yang, C., Zhang, Y., Liu, Z., and Sun, M. (2024). Iterative experience refinement of software-developing agents. *arXiv preprint arXiv:2405.04219*.
- Qiao, B., Li, L., Zhang, X., He, S., Kang, Y., Zhang, C., Yang, F., Dong, H., Zhang, J., Wang, L., et al. (2023). Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*.
- Qu, C., Dai, S., Wei, X., Cai, H., Wang, S., Yin, D., Xu, J., and Wen, J.-R. (2024). Tool learning with large language models: A survey. *arXiv preprint arXiv:2405.17935*.
- 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. *Journal of Machine Learning Research*, 25(209):1–6.
- Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- SAS Institute Inc. (2015). Sas/stat® 14.1 user’s guide.

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., Rozière, B., Goyal, N., Hambro, E., Azhar, F., 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. *International Journal of Data Science and Analytics*, 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 Biology*, 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., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. (2023). Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Yang, Z., Zhou, Z., Wang, S., Cong, X., Han, X., Yan, Y., Liu, Z., Tan, Z., Liu, P., Yu, D., Liu, Z., Shi, X., and Sun, M. (2024). Matplotagent: Method and evaluation for llm-based agentic scientific data visualization.

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

Zeng, A., Liu, X., Du, Z., Wang, Z., Lai, H., Ding, M., Yang, Z., Xu, Y., Zheng, W., Xia, X., et al. (2022). Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.

- 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*.
- Zhang, W., Shen, Y., Lu, W., and Zhuang, Y. (2023). Data-copilot: Bridging billions of data and humans with autonomous workflow. *arXiv preprint arXiv:2306.07209*.
- 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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.
- Zhou, A., Wang, K., Lu, Z., Shi, W., Luo, S., Qin, Z., Lu, S., Jia, A., Song, L., Zhan, M., and Li, H. (2023a). Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification. *arXiv preprint arXiv:2308.07921*.
- Zhou, W., Jiang, Y. E., Li, L., Wu, J., Wang, T., Qiu, S., Zhang, J., Chen, J., Wu, R., Wang, S., Zhu, S., Chen, J., Zhang, W., Tang, X., Zhang, N., Chen, H., Cui, P., and Sachan, M. (2023b). Agents: An open-source framework for autonomous language agents. *arXiv preprint arXiv:2309.07870*.

Supplementary Materials

The Supplementary Materials include methodological details, experimental data, case studies and experimental setting. Specifically, we present our initial idea of a function-calling-based agent system in Section A. In Section B, we describe the core modules involved in kernel development and the datasets used in our experiments. Additionally, Section C presents several case studies demonstrating the use of LAMBDA, including data analysis, a self-correcting mechanism, the integration of human intelligence, its application in education, and report generation. Finally, we outline our experimental setup in Section D.

A Function Calling Based Agent System

The first idea that came to our mind was function calling. We developed extensive APIs that encompass a wide range of data processing and machine learning functionalities, including statistical descriptions (e.g., mean, median, standard deviation), encoding schemes (e.g., one-hot encoding, ordinal encoding), data partitioning, and model training (e.g., logistic regression, decision tree). We utilized five function libraries to build these APIs, each tailored for different purposes: the Data Description Library, Data Visualization Library, Data Processing Library, Modeling Library, and Evaluation Library. Each library caches variables such as processed data and models throughout the program’s lifecycle. The framework and workflow are illustrated in Figure S.1.

We implemented the function calling service by ReAct. Specifically, when prompted to generate text up to the “Observation” section, the LLM should halt generation at this point. This is essential as the “Observation” section requires the outcome of API execution to prevent LLMs from generating results autonomously. The details are depicted in Figure S.2.

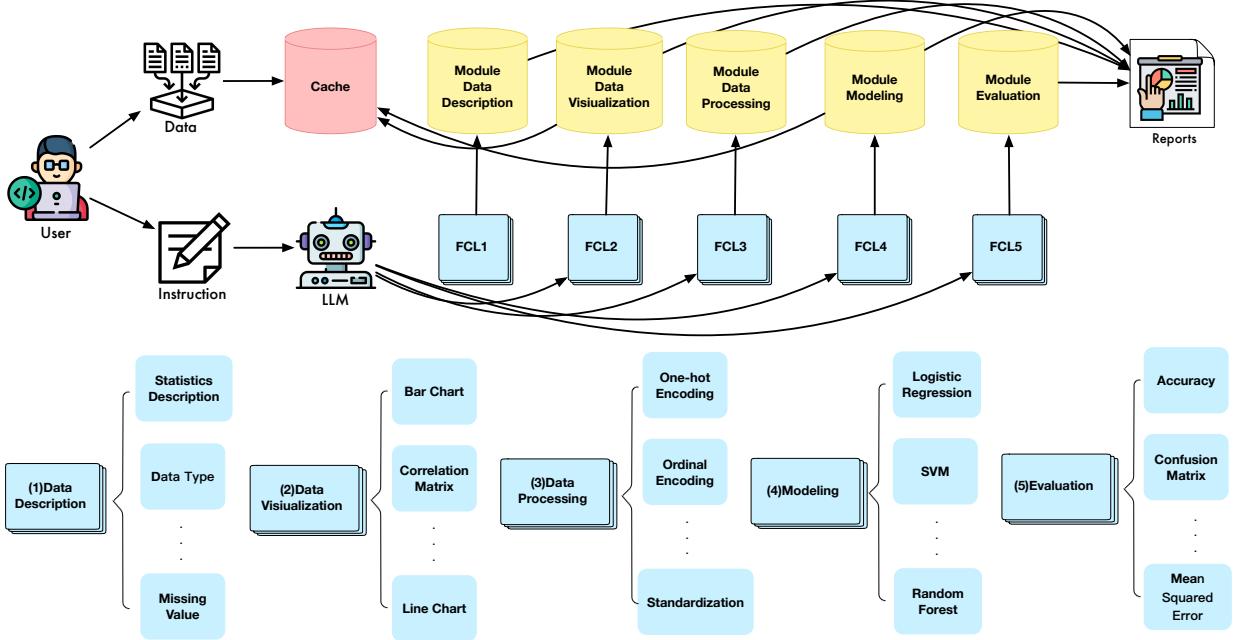


Figure S.1: Agent system design by the function calling method. The FCL means function calling library.

B Kernel

The `CodeKernel` is designed to facilitate the execution of code within a Jupyter Notebook environment. It interacts with the Jupyter backend kernel to manage code execution, handle errors, and oversee the lifecycle of the kernel. This class provides an interface for executing code in a controlled manner, ensuring that outputs are captured and processed effectively.

B.1 Constructor

The constructor of the `CodeKernel` class initializes an instance of the kernel `kernel_name`, `kernel_id`, `kernel_config_path`, `python_path`, `ipython_path`, `init_file_path`, and `verbose`. The initialization process involves setting up environment variables based on the specified paths for Python or IPython, which are crucial for ensuring the correct execution context.

The kernel is initialized using the `KernelManager` from the `jupyter_client` library, allowing for granular control over the kernel's operations. Depending on whether a configuration file is provided, the kernel is either started with the given settings or defaults

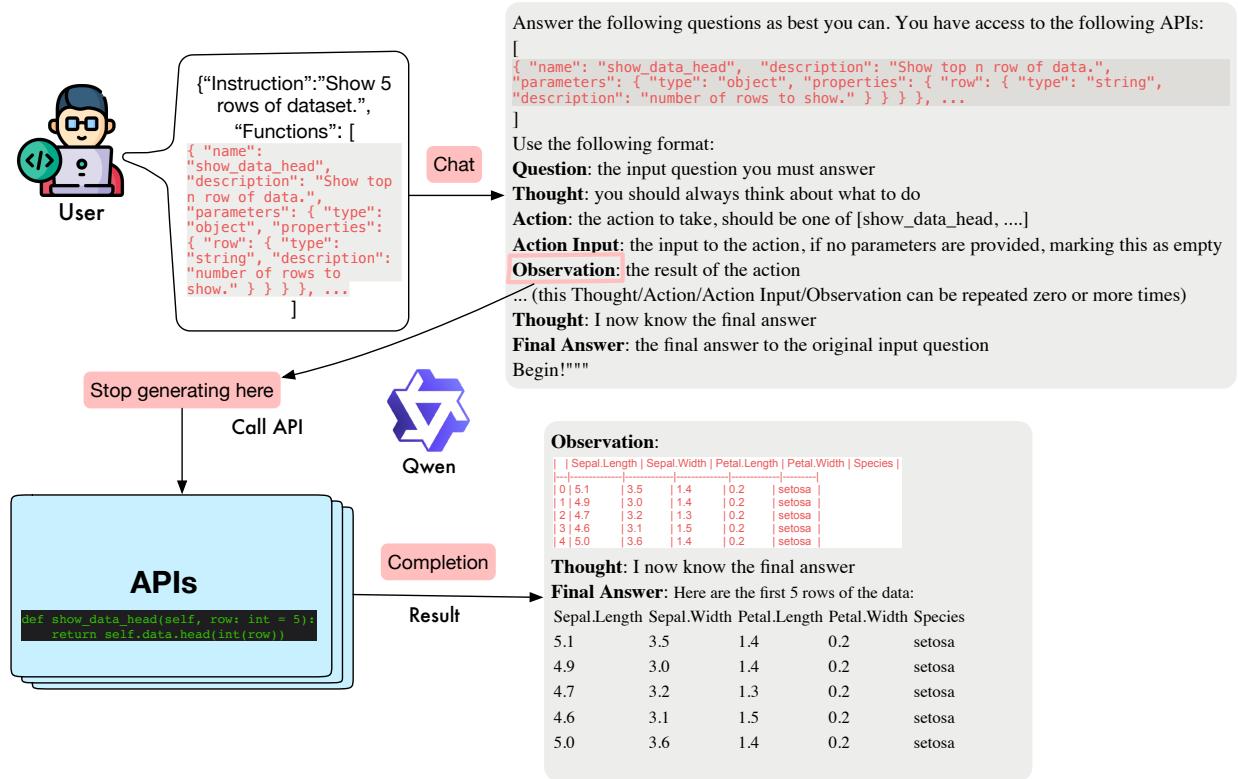


Figure S.2: Workflow of function calling service, demonstrated by Qwen1.5 and ReAct.

are applied. The constructor also provides detailed logging of the kernel’s connection information if verbosity is enabled.

B.2 Code execution

The `execute` method is the primary mechanism for running code within the initialized kernel environment. This method sends a code string to the kernel for execution, utilizing a blocking client to ensure that the process completes before continuing. The method retrieves both shell messages and IOPub messages, which contain critical information about the execution status and outputs.

Outputs are processed to handle standard output (`stdout`), errors (`stderr`), and other response types from the kernel. The method returns a tuple consisting of the shell message and a list of processed output messages. This design allows for comprehensive handling of multi-line outputs and ensures that the relevant results are captured and returned.

B.3 Interactive execution

The `execute_interactive` method facilitates the execution of code in an interactive manner, where outputs are immediately accessible to the user. This method ensures that the code execution is monitored closely, with specific handling for timeout scenarios. If the `verbose` flag is set, the method provides detailed output about the execution process, aiding in debugging and analysis.

B.4 Code inspection

The `inspect` method allows for the introspection of code by sending it to the kernel and retrieving detailed information about variables, functions, and other elements within the code. This method is particularly useful for debugging, as it provides real-time insights into the structure and behavior of the code being executed. The inspection results are returned as part of the shell message, which can be further processed or displayed.

B.5 Error handling

The `get_error_msg` and `check_msg` methods are responsible for handling errors that occur during code execution. The `get_error_msg` method extracts detailed error messages from the kernel's response, ensuring that these messages are accessible for debugging purposes. The `check_msg` method evaluates the status of the execution and prints error traces if any issues are detected, providing a clear indication of what went wrong during the execution.

B.6 Kernel management

The `CodeKernel` class includes several methods for managing the lifecycle of the kernel:

- `shutdown`: This method stops both the backend kernel and the associated code kernel, ensuring that all resources are released.
- `restart`: This method restarts the kernel, providing a clean slate for subsequent code executions.

- **start**: This method initializes the code kernel if it is not already running, allowing for new executions.
- **interrupt**: This method interrupts a long-running or unresponsive kernel, providing control over runaway processes.
- **is_alive**: This method checks whether the kernel is active and responsive, offering a way to monitor the kernel's status.

B.7 Datasets

Here we give the information on the sources of the datasets used in Section 4.3.

Table S.1: Datasets used in this study.

DataSets	Usage
AIDS Clinical Trials Group Study 175 ¹	Classification
NHANES ²	Classification
Breast Cancer Wisconsin ³	Classification
Wine ⁴	Classification
Concrete Compressive Strength ⁵	Regression
Combined Cycle Power Plant ⁶	Regression
Abalone ⁷	Regression - Education Case Study
Airfoil Self-Noise ⁸	Regression
Iris ⁹	Classification - Data Analysis Case Study
Heart Disease ¹⁰	Regression - Education Case Study

¹ <https://archive.ics.uci.edu/dataset/890/aids+clinical+trials+group+study+175>

² [https://archive.ics.uci.edu/dataset/887/national+health+and+nutrition+health+survey+2013-2014+\(nhanes\)+age+prediction+subset](https://archive.ics.uci.edu/dataset/887/national+health+and+nutrition+health+survey+2013-2014+(nhanes)+age+prediction+subset)

³ <https://archive.ics.uci.edu/dataset/891/cdc+diabetes+health+indicators>

⁴ <https://archive.ics.uci.edu/dataset/109/wine>

⁵ <https://archive.ics.uci.edu/dataset/165/concrete+compressive+strength>

⁶ <https://archive.ics.uci.edu/dataset/294/combined+cycle+power+plant>

⁷ <https://archive.ics.uci.edu/dataset/1/abalone>

⁸ <https://archive.ics.uci.edu/dataset/291/airfoil+self+noise>

⁹ <https://archive.ics.uci.edu/dataset/53/iris>

¹⁰ <https://archive.ics.uci.edu/dataset/45/heart+disease>

C Case studies

This section presents several case studies demonstrating the use of LAMBDA, including data analysis, a self-correcting mechanism, the integration of human intelligence, its application in education, and report generation.

Data Analysis (Iris Dataset)

HUMAN	LAMBDA
<p>1 Encoding the target `\"Species\" to number like 0,1,2</p> <pre>'''python import pandas as pd # Load the dataset data = pd.read_csv('/Users/stephensun/Desktop/pypro/dsagent_ci/cache/conv_cache/2c23cf4b-31a8-4aeb-8689-13b9fbdfb7fc-2024-04-29/iris.csv') # Encode the target variable from sklearn.preprocessing import LabelEncoder label = LabelEncoder() data['Species'] = label.fit_transform(data['Species']) # Display the encoded values data['Species'].head()</pre> <p>Execution result:</p> <pre>0.0 1.0 2.0 3.0 4.0</pre> <p>The target variable `\"Species\" has been successfully encoded to numerical values 0, 1, and 2. This encoding will facilitate the use of the variable in various machine learning algorithms and analyses. Now that the target variable is encoded, you can proceed with further data preprocessing, model building, or analysis as needed."</p>	
<p>2 Draw a correlation matrix</p> <pre>'''python import seaborn as sns import matplotlib.pyplot as plt # Create a correlation matrix correlation_matrix = data.corr() # Plot the correlation matrix plt.figure(figsize=(8, 6)) sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', cbar=True) plt.title('Correlation Matrix') plt.savefig('/Users/stephensun/Desktop/pypro/dsagent_ci/cache/conv_cache/2c23cf4b-31a8-4aeb-8689-13b9fbdfb7fc-2024-04-29/correlation_matrix.png') Execution result:</pre> <p>The correlation matrix has been successfully plotted using a heatmap. The heatmap visually represents the correlation coefficients between different features in the dataset. This information can be useful for understanding the relationships between the variables and identifying potential multicollinearity.</p> <p>The resulting correlation matrix plot has been saved as a PNG file, and it provides a clear visualization of the correlations between the features in the dataset. Further analysis or modeling can be informed by the insights gained from this correlation matrix.</p>	
<p>3 Split the data to train and test with a proportion of 8:2</p> <pre>'''python from sklearn.model_selection import train_test_split # Separate the features and the target variable X = data.drop('Species', axis=1) y = data['Species'] # Split the data into training and testing sets X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) Execution result:</pre> <p>The dataset has been split into training and testing sets with a proportion of 80% for training data and 20% for testing data. This is a common split ratio for training and evaluating machine learning models. The training set (X_train and y_train) can be used to train the model, while the testing set (X_test and y_test) can be used to evaluate the model's performance and generalization to new data.</p> <p>You can now use the X_train, X_test, y_train, and y_test datasets for training and evaluating machine learning models."</p>	
<p>4 Train a logistic regression model and evaluate it by accuracy on the test set.</p> <pre>'''python from sklearn.linear_model import LogisticRegression from sklearn.metrics import accuracy_score # Initialize the logistic regression model logreg_model = LogisticRegression(max_iter=1000) # Train the model on the training data logreg_model.fit(X_train, y_train) # Make predictions on the test data y_pred = logreg_model.predict(X_test) # Evaluate the model using accuracy accuracy = accuracy_score(y_test, y_pred) accuracy Execution result: 1.0</pre> <p>The logistic regression model has been trained and evaluated on the test set, yielding an accuracy of 1.0 or 100%. An accuracy of 1.0 indicates that the model predicted the classes of all samples in the test set perfectly.</p> <p>While achieving a high accuracy is desirable, it is important to consider the possibility of overfitting. In this case, the high accuracy on the test set may indicate overfitting. It's recommended to further evaluate the model's performance using techniques such as cross-validation, precision-recall analysis, or confusion matrix to gain a deeper understanding of its predictive capabilities.</p> <p>Additionally, given the small size of the Iris dataset, a high accuracy on the test set is not unexpected. It would be advisable to use more complex datasets for a more realistic assessment of model performance."</p>	

Figure S.3: Case study: Analysis of Iris dataset.

Data Analysis (Iris Dataset)

HUMAN	LAMBDA
5 Train a svm model and evaluate it by accuracy on the test set.	<pre>'''python from sklearn.svm import SVC # Initialize the SVM model svm_model = SVC(kernel='linear') # Train the model on the training data svm_model.fit(X_train, y_train) # Make predictions on the test data y_pred_svm = svm_model.predict(X_test) # Evaluate the model using accuracy accuracy_svm = accuracy_score(y_test, y_pred_svm) accuracy_svm ''' Execution result: 1.0 The support vector machine (SVM) model with a linear kernel has been trained and evaluated on the test set, yielding an accuracy of 1.0 or 100%. Similar to the logistic regression model, an accuracy of 1.0 may indicate overfitting. As mentioned previously, it's advised to further evaluate the model's performance using additional techniques and metrics such as cross-validation, precision-recall analysis, or confusion matrix to gain a more comprehensive understanding of its predictive capabilities. While high accuracy is desirable, it is important to maintain a critical perspective and consider the potential for overfitting in the context of the specific dataset and model evaluation.'</pre>
6 Train a mlp model and evaluate it by accuracy on the test set.	<pre>'''python from sklearn.neural_network import MLPClassifier # Initialize the MLP model mlp_model = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=500) # Train the model on the training data mlp_model.fit(X_train, y_train) # Make predictions on the test data y_pred_mlp = mlp_model.predict(X_test) # Evaluate the model using accuracy accuracy_mlp = accuracy_score(y_test, y_pred_mlp) accuracy_mlp ''' Execution result: 1.0 The multi-layer perceptron (MLP) model has been trained and evaluated on the test set, resulting in an accuracy of 1.0 or 100%. As with the previous models, achieving an accuracy of 1.0 may indicate potential overfitting. In practice, it is important to critically assess the model's performance using a variety of evaluation metrics and techniques, as well as cross-validation, to ensure a robust assessment. It's worth considering other evaluation metrics such as precision, recall, and F1 score to gain a more comprehensive understanding of the model's performance characteristics. Additionally, exploring techniques for optimizing the model's architecture, regularization, and hyperparameters could further enhance its generalization capabilities.'</pre>
7 Train a decision tree model and evaluate it by accuracy on the test set.	<pre>'''python from sklearn.tree import DecisionTreeClassifier # Initialize the decision tree model dt_model = DecisionTreeClassifier() # Train the model on the training data dt_model.fit(X_train, y_train) # Make predictions on the test data y_pred_dt = dt_model.predict(X_test) # Evaluate the model using accuracy accuracy_dt = accuracy_score(y_test, y_pred_dt)\naccuracy_dt ''' Execution result: 1.0 The decision tree model has been trained and evaluated on the test set, resulting in an accuracy of 1.0 or 100%. It's important to consider that achieving a high accuracy score on a small and well-structured dataset like the Iris dataset may indicate overfitting. Given the perfect accuracy, it's recommended to further evaluate the model's performance using additional techniques such as cross-validation, precision-recall analysis, or confusion matrix to gain a more comprehensive understanding of its predictive capabilities. It's also important to take into account the interpretability of the decision tree model and how it aligns with the specific requirements of the analysis or application. Further model tuning and exploration of ensemble methods may also be beneficial for improving generalization and ensuring robust performance on new data.'</pre>
8 train a random forest model and evaluate it by accuracy on the test set.	<pre>'''python from sklearn.ensemble import RandomForestClassifier # Initialize the random forest model rf_model = RandomForestClassifier(n_estimators=100, random_state=42) # Train the model on the training data rf_model.fit(X_train, y_train) # Make predictions on the test data y_pred_rf = rf_model.predict(X_test) # Evaluate the model using accuracy accuracy_rf = accuracy_score(y_test, y_pred_rf) accuracy_rf ''' Execution result: 1.0 The random forest model has been trained and evaluated on the test set, resulting in an accuracy of 1.0 or 100%. As with previous models, it's important to remain cautious about the potential for overfitting when achieving a perfect accuracy score, especially on a small dataset like Iris. While a high accuracy score is desirable, it's essential to consider the model's generalization to new data. Additional evaluation metrics, cross-validation, and further analysis of model performance can provide a more comprehensive understanding of the random forest model's capabilities. In addition to accuracy, it would be beneficial to explore other evaluation metrics and techniques such as precision, recall, and F1 score, as well as feature importance analysis, to gain insights into the model's predictive performance and behavior.'</pre>

Figure S.4: Cont. Case study: Analysis of Iris dataset.

Data Analysis (Iris Dataset)

HUMAN	LAMBDA
9 Use GridSearchCV to find the best hyperparameters of mlp in a parameters group. you can define the search group by yourself, 5 group is enough.	<pre> ```python from sklearn.model_selection import GridSearchCV # Define the parameter grid for the MLP model param_grid_mlp = [{ 'hidden_layer_sizes': [(50), (100), (50, 50), (100, 50), (100, 100)], 'activation': ['logistic', 'tanh', 'relu'], 'solver': ['adam', 'sgd'], 'max_iter': [200, 500, 1000] }] # Initialize the MLP model mlp_model = MLPClassifier() # Initialize and fit the GridSearchCV grid_search_mlp = GridSearchCV(mlp_model, param_grid_mlp, cv=5, scoring='accuracy', n_jobs=-1) grid_search_mlp.fit(X_train, y_train) # Get the best parameters and the best accuracy score best_params_mlp = grid_search_mlp.best_params_ best_score_mlp = grid_search_mlp.best_score_ ``` Execution result: `{'activation': 'tanh', 'hidden_layer_sizes': (100,), 'solver': 'sgd', 'max_iter': 200}, 0.975` The best hyperparameters for the MLP model identified through GridSearchCV are as follows: - Activation function: 'tanh' - Hidden layer sizes: (100,) - Maximum number of iterations: 200 - Solver: 'sgd' The best accuracy score achieved using these hyperparameters is 0.975. This grid search process provides valuable insights into the combination of hyperparameters that optimizes the MLP model's performance on the training data. Using the best hyperparameters identified, you can retrain the MLP model and evaluate its performance on the test set to ascertain its generalization and predictive capabilities. Additionally, a more in-depth analysis of the model's performance, such as precision-recall analysis and model interpretation, can provide further insights into its behavior and characteristics." </pre>
10 Good, save all models before	<pre> ```python import joblib # Save the trained models model_files = { 'logistic_regression_model.pkl': logreg_model, 'svm_model.pkl': svm_model, 'mlp_model.pkl': mlp_model, 'decision_tree_model.pkl': dt_model, 'random_forest_model.pkl': rf_model } for file_name, model in model_files.items(): joblib.dump(model, file_name) # Save the best MLP model from GridSearchCV joblib.dump(grid_search_mlp.best_estimator_, 'best_mlp_model.pkl') ``` Execution result: `logistic_regression_model.pkl svm_model.pkl decision_tree_model.pkl random_forest_model.pkl best_mlp_model.pkl` The trained models have been successfully saved. Additionally, the best MLP model resulting from the GridSearchCV process has been saved as 'best_mlp_model.pkl'. These saved model files can be used for future predictions, analysis, or deployment without the need to retrain the models from scratch. If you need to transfer these models to another location or share them with others, you can download the files and store them in a location of your choice.\n\nIf you have any further questions or need additional assistance, feel free to ask! decision_tree_model.pkl best_mlp_model.pkl logistic_regression_model.pkl svm_model.pkl random_forest_model.pkl mlp_model.pkl </pre>

Figure S.5: Cont. Case study: Analysis of Iris dataset.

Self-correcting Mechanism

Instruction: Draw a confusion matrix

Programmer response [Code]

```
import seaborn as sns
import matplotlib.pyplot as plt
# Calculate the correlation matrix
correlation_matrix = data.corr()
# Create a heatmap of the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.savefig('correlation_matrix.png')
```

1

Execution message [Error message]

2

```
ValueError
Cell In[4], line 5
    2 import matplotlib.pyplot as plt
    4 # Calculate the correlation matrix
--> 5 correlation_matrix = data.corr()
    7 # Create a heatmap of the correlation matrix
    8 plt.figure(figsize=(8, 6))
.....
ValueError: could not convert string to float: 'setosa'
```

Inspector prompt

3

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.

- bug code:

[Code]

When executing 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.

Modification method:

Inspector response [Fix_method]

4

The error message "could not convert string to float: 'setosa'" indicates that there are non-numeric values present in the data that is causing the problem. To resolve this issue, you can consider the following modification method: Check the columns of the data and identify if there are any non-numeric columns that are causing the problem. It's possible that the 'species' column in the 'iris' dataset contains non-numeric values.

You can either remove the non-numeric columns or convert them into numeric format if they contain categorical data. For example, you can use one-hot encoding or label encoding to convert categorical data into numeric format.

Once the non-numeric values are processed, the correlation matrix should be computed without any issues.

Repair prompt for programmer

5

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.

- bug code: [Code]

When executing above code, errors occurred:

[Error message]

Please check the implementation of the function and fix the code based on the modification method.

- modification method:

[Fix_method]

The code you modified (should be wrapped in ``python``):

Programmer response

6

```
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
Check the data to identify non-numeric columns
print(data.dtypes)
Convert categorical data to numeric format using one-hot encoding
data = pd.get_dummies(data, columns=['Species'], drop_first=True)
Draw a correlation matrix after processing non-numeric data
correlation_matrix = data.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.savefig('correlation_matrix.png')
```
```

Execution result

7

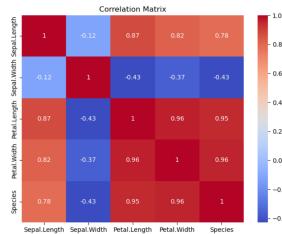


Figure S.6: Case study: An example of self-correcting mechanism in LAMBDA.

Knowledge Integration

Nearest Correlation Matrix

Paper: Houduo Qi and Defeng Sun. A quadratically convergent newton method for computing the nearest correlation matrix. SIAM Journal on Matrix Analysis and Applications, 28(2):360–385, 2006

Python code: <https://www.polyu.edu.hk/ama/profile/dfsun/CorrelationMatrix.py>

Knowledge Base - Interface

```
def NearestCorrelationMatrix( g_input, b_input=None, tau=None, tol=None):
    print('-- Semismooth Newton-CG method starts -- \n')
    ....
    return x_result, y

def my_gradient( y_input, lamb, p_input, b_0, n):
    f = 0.0
    ....
    return f, Fy

def my_issorted( x_input, flag):
    n = x_input.size
    ....
    return tf_value

def my_jacobian_matrix( x, omega_12, p_input, n):
    x_result = np.zeros((n, 1))
    ....
    return x_result

def my_mex eig( x_input):
    [n, m] = x_input.shape
    ....
    return p_x, lamb

def my_omega_mat( p_input, lamb, n):
    idx_idp = np.where(lamb > 0)
    ....
    return omega_12

def my_pca( x_input, lamb, p_input, b_0, n):
    x_pca = x_input
    ....
    return x_pca

def my_pre_cg( b, tol, maxit, c, omega_12, p_input, n):
    # Initializations
    ....
    return p, flag, relres, iterk

def my_precond_matrix( omega_12, p_input, n):
    [r, s] = omega_12.shape
    ....
    return c

def test_ncm():
    # test
    n = 3000
    data_g_test = scipy.randn(n, n)
    data_g_test = (data_g_test + data_g_test.transpose()) / 2.0
    data_g_test = data_g_test - np.diag(np.diag(data_g_test)) + np.eye(n)
    b = np.ones((n, 1))
    tau = 0
    tol = 1.0e-6
    [x_test_result, y_test_result] = NearestCorrelationMatrix(data_g_test, b, tau, tol)
    print("The x_test_result: \n", x_test_result)
    print()
    print("The y_test_result: \n", y_test_result)
```

Figure S.7: Case study: An example of integrating human intelligence in the analysis. The red part is the ‘core’ function in the ‘Core’ mode.⁴⁵

Knowledge Integration

| HUMAN | LAMBDA |
|--|---|
| <p>I want to calculate the nearest correlation matrix by the Quadratically Convergent Newton Method. Please write a well-detailed code. The code gives details of the computation for each iteration, such as the norm of gradient, relative duality gap, dual objective function value, primal objective function value, and the running time.</p> <p>Using the following parameters to run a test case and show the result:</p> <p>Set a 2000x2000 random matrix whose elements are randomly drawn from a standard normal distribution, the matrix should be symmetric positive, and semi-definite.</p> <p>Set the b vector by 2000x1 with all elements 1.</p> <p>Set tau by 0.1, and tolerance error by 1.0e-7.</p> | <p>Retrieval: The retriever find the following pieces of code cloud address the problem.
Retrieval code:
[Interface:
def NearestCorrelationMatrix(g_input, b_input=None, tau=None, tol=None):
 print('-- Semismooth Newton-CG method starts --\n')

 return x_result, y
....
]

Here is the modified code to run a test case using the provided parameters:
```python
import numpy as np
import scipy.linalg as la

Set the parameters
n = 2000
g_input = np.random.randn(n, n)
g_input = (g_input + g_input.transpose()) / 2.0
g_input = g_input - np.diag(np.diag(g_input)) + np.eye(n)
b_input = np.ones((n, 1))
tau = 0.1
tol = 1.0e-7

Run the test case
x_test_result, y_test_result = NearestCorrelationMatrix(g_input, b_input, tau, tol)

print("The x_test_result: \n", x_test_result)
print()
print("The y_test_result: \n", y_test_result)
```
This code sets the parameters n, g_input, b_input, tau, and tol according to the problem statement. It then calls the NearestCorrelationMatrix function with these parameters and prints the results.

Note that I didn't modify the NearestCorrelationMatrix function itself, as it seems to be a complex implementation of the Semismooth Newton-CG method. I only provided the test case code to run the function with the given parameters.

Execute result:
-- Semismooth Newton-CG method starts --

Newton-CG: Initial Dual objective function value: [[475528.24357215]]
Newton-CG: Initial Primal objective function value: 933281.3129374245
Newton-CG: Norm of Gradient: 579.6794822761284
Newton-CG: computing time used so far: 10.611207962036133
Newton-CG: Number of CG Iterations== 1
.....
Newton-CG: computing time for eigenvalue decompositions: ====== 42.30057191848755
Newton-CG: computing time used for equal weight calibration ====== 47.19361400604248

The x_test_result:
[[1. -0.03941701 0.03307246 ... -0.01731567 -0.11538529
-0.01449621]
[-0.03941701 1. -0.02205682 ... 0.14064247 0.01924966
-0.03100769]
[0.03307246 -0.02205682 1. ... 0.03770609 0.04565391
0.00780341]
...
[-0.01731567 0.14064247 0.03770609 ... 1. -0.04084723
0.04516027]
[-0.11538529 0.01924966 0.04565391 ... -0.04084723 1.
0.03888915]
[-0.01449621 -0.03100769 0.00780341 ... 0.04516027 0.03888915]]

The y_test_result:
[[-46.71400534]
[-41.15359542]
[-44.00442815]
...
[-41.00521091]
[-46.99753955]
[-39.80357383]]

Based on the executing result, the Semismooth Newton-CG method converged to a solution with a relative duality gap of 3.15279506e-08. The final optimal solution has a rank of 214. The computation time used for equal weight calibration is 47.19361400604248 seconds.</p> |

Figure S.8: Cont. Case study: An example of integrating human intelligence in the analysis.

Education (Lecture and Homework Design)

Teacher View

I am a teacher. I will teach lasso in next class, totally 2 hours. You should write a syllabus for me. Besides, you should also design an experiment task for students to learn Lasso. Use the datasets like Abalone: <https://archive.ics.uci.edu/dataset/1/abalone>

LAMBDA

Course Outline:

Title: Introduction to Lasso Regression

Duration: 2 hours

I. Introduction to Lasso Regression

- A. Overview of regression analysis
- B. Introduction to Lasso regression
- C. Advantages and disadvantages of Lasso regression

II. Understanding Lasso Regression

- A. Understanding the L1 regularization
- B. Comparison with traditional regression techniques
- C. Use cases and applications

III. Implementing Lasso Regression

- A. Data preprocessing for Lasso regression
- B. Training Lasso regression model
- C. Evaluating model performance

IV. Advanced Topics

- A. Hyperparameter tuning
- B. Cross-validation for model selection
- C. Dealing with multicollinearity

V. Case Studies and Practical Applications

- A. Real-world examples of Lasso regression
- B. Discussion on potential research areas

Experiment Task:

Task: Prediction of Abalone Age using Lasso Regression

Objective: The objective of this experiment is to apply Lasso regression on the Abalone dataset to predict the age of abalone based on various physical measurements.

Dataset: Abalone dataset can be downloaded from <https://archive.ics.uci.edu/dataset/1/abalone> or any other dataset containing physical measurements of abalone and their ages.

Steps:

Data Preprocessing:

Load the dataset and inspect the features and target variable

Perform any necessary data cleaning or preprocessing

Exploratory Data Analysis:

Visualize the relationship between the features and target variable

Analyze correlations and distributions of the variables

Implementing Lasso Regression:

Split the dataset into training and testing sets

Implement Lasso regression using a suitable library (e.g., scikit-learn in Python)

Train the model on the training set and evaluate its performance on the testing set

Model Evaluation:

Evaluate the performance of the Lasso regression model using appropriate metrics (e.g., mean squared error, R-squared)

Advanced Analysis:

Experiment with different values of alpha (regularization parameter) to understand its impact on the model

Perform cross-validation to select the best model

Homework:

Students will be tasked with writing a summary of the key concepts and advantages of Lasso regression compared to traditional regression techniques.

Students will apply Lasso regression to a different dataset of their choice and write a report on their findings, including model performance and insights gained from the analysis.

Figure S.9: Case study: LAMBDA in education. We use teaching Lasso as an example.

Education (Do Homework on LAMBDA)

STUDENT

LAMBDA

- 1 This is Abalone dataset. I want to predict the age of Abalone, corresponding column is Ring. First, show 5 rows of the dataset.

```
```
python
Load the Abalone dataset
import pandas as pd
data = pd.read_csv('/Users/stephensun/Desktop/pypro/LAMBDA/cache/conv_cache/4e7d8506-6234-44b3-a177-aed15f488027-2024-05-20/Abalone.csv')

Show 5 rows of the dataset
data.head()
```

Execute result:
Here is the rewritten table in markdown format:
```

| Sex | Length | Diameter | Height | Whole_weight | Shucked_weight | Viscera_weight | Shell_weight | Rings |
|-----|--------|----------|--------|--------------|----------------|----------------|--------------|-------|
| M | 0.455 | 0.365 | 0.095 | 0.5140 | 0.2245 | 0.1010 | 0.150 | 15 |
| M | 0.350 | 0.265 | 0.090 | 0.2255 | 0.0995 | 0.0485 | 0.070 | 7 |
| F | 0.530 | 0.420 | 0.135 | 0.6770 | 0.2565 | 0.1415 | 0.210 | 9 |
| M | 0.440 | 0.365 | 0.125 | 0.5160 | 0.2155 | 0.1140 | 0.155 | 10 |
| I | 0.330 | 0.255 | 0.080 | 0.2050 | 0.0895 | 0.0395 | 0.055 | 7 |

The dataset includes information about the sex, physical dimensions, and number of rings for abalones. The 'Rings' column can be used to predict the age of the abalones, as it represents the number of rings on the abalone shell. The next step would be to perform exploratory data analysis to understand the data distribution and relationships between variables. Additionally, feature engineering and model selection for age prediction can be explored.

- 2 Encoding column Sex as one-hot vectors and replace original column.

```
```
python
Perform one-hot encoding on the 'Sex' column
data = pd.get_dummies(data, columns=['Sex'], drop_first=True)

Display the first 5 rows of the updated dataset
data.head()
```

Execute result:
Here is the rewritten table in markdown format:
```

| Length | Diameter | Height | Whole_weight | Shucked_weight | Viscera_weight | Shell_weight | Rings | Sex_I | Sex_M |
|--------|----------|--------|--------------|----------------|----------------|--------------|-------|-------|-------|
| 0.455 | 0.365 | 0.095 | 0.5140 | 0.2245 | 0.1010 | 0.150 | 15 | False | True |
| 0.350 | 0.265 | 0.090 | 0.2255 | 0.0995 | 0.0485 | 0.070 | 7 | False | True |
| 0.530 | 0.420 | 0.135 | 0.6770 | 0.2565 | 0.1415 | 0.210 | 9 | False | False |
| 0.440 | 0.365 | 0.125 | 0.5160 | 0.2155 | 0.1140 | 0.155 | 10 | False | True |
| 0.330 | 0.255 | 0.080 | 0.2050 | 0.0895 | 0.0395 | 0.055 | 7 | True | False |

The 'Sex' column has been successfully encoded into one-hot vectors, replacing the original column with new columns 'Sex_I' and 'Sex_M'. This transformation will enable the machine learning models to interpret the 'Sex' attribute effectively. The next step would involve splitting the data into features and labels and proceeding with the model training process to predict the age of the abalones.

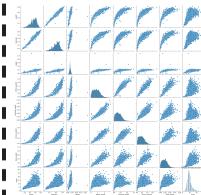
- 3 Visualize the relationship between the features and target variable. Save the figure finally.

```
```
python
import seaborn as sns
import matplotlib.pyplot as plt

Plot the pairplot for the relationship between features and target variable
sns.pairplot(data, vars=['Length', 'Diameter', 'Height', 'Whole_weight', 'Shucked_weight', 'Viscera_weight', 'Shell_weight', 'Rings'])
plt.title('Pairplot of Features and Rings')
plt.savefig('/Users/stephensun/Desktop/pypro/LAMBDA/cache/conv_cache/4e7d8506-6234-44b3-a177-aed15f488027-2024-05-20/abalone_pairplot.png')

Execute result:
/Users/stephensun/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
self._figure.tight_layout(*args, **kwargs)
```

The pairplot illustrates the pairwise relationships between the features and the target variable 'Rings'. It provides insights into the correlations and distribution of the features. The next steps could involve feature scaling, splitting the data into training and testing sets, and then training a predictive model to predict the age of the abalones.
```



- 4 Split the dataset into training and testing sets with a proportion of 8:2. Show the shape of the training and testing sets.

```
```
python
from sklearn.model_selection import train_test_split

Split the dataset into training and testing sets
X = data.drop(['Rings'], axis=1) # Features
y = data['Rings'] # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Display the shape of the training and testing sets
print('Training set shape:', X_train.shape, y_train.shape)
print('Testing set shape:', X_test.shape, y_test.shape)
```

Execute result:
Training set shape: (3341, 9) (3341,)
Testing set shape: (836, 9) (836,)

Training set shape: (3341, 9) (3341,)
Testing set shape: (836, 9) (836,)

The dataset has been successfully split into training and testing sets with a ratio of 80:20, resulting in 3341 samples in the training set and 836 samples in the testing set, with 9 features in each set. The next step would involve model training using the training set, followed by evaluating the model's performance on the testing set to predict the age of the abalones.
```

Figure S.10: Cont. Case Study: LAMBDA in education. Students complete assignment.

Education (Do Homework on LAMBDA)

| STUDENT | LAMBDA |
|---|---|
| <p>5 Implement Lasso regression using a suitable library (e.g., scikit-learn in Python) and train the lasso on the training set and evaluate its MSE on the testing set.</p> | <pre>python from sklearn.linear_model import Lasso from sklearn.metrics import mean_squared_error # Initialize the Lasso regression model lasso_model = Lasso(alpha=0.1) # Train the Lasso model on the training set lasso_model.fit(X_train, y_train) # Make predictions using the trained model y_pred = lasso_model.predict(X_test) # Calculate Mean Squared Error (MSE) on the testing set mse = mean_squared_error(y_test, y_pred) mse ``` Execute result:
 7.2185794133162995</pre> <p>The Mean Squared Error (MSE) of the Lasso regression model on the testing set is 7.22. This value represents the average squared difference between the actual and predicted age of the abalones. The next step could involve further model evaluation, hyperparameter tuning, and exploring alternative regression models to improve performance.</p> |
| <p>6 Use grid_search_cv to perform cross-validation to select the best model with different values of alpha. Select alpha in [0.1, 0.5, 1, 1.5, 2]. Print the best parameters and best score (MSE). Set n_jobs=-1 and random_state=42. Save the best model finally.</p> | <pre>python import matplotlib.pyplot as plt # Extract alpha values and corresponding MSE from the GridSearchCV results alphas = [0.1, 0.5, 1, 1.5, 2] mse_values = [-result for result in grid_search_cv_results_['mean_test_score']] # Create a line plot to show the relationship between alpha and MSE plt.figure(figsize=(10, 6)) plt.plot(alphas, mse_values, marker='o', linestyle='-', color='b') plt.title('Alpha vs. Mean Squared Error (MSE)') plt.xlabel('Alpha') plt.ylabel('Mean Squared Error (MSE)') plt.xticks(alphas) plt.grid(True) # Save the figure plt.savefig('/Users/stephensun/Desktop/pypro/LAMBDA/cache/conv_cache/4e7d8506-6234-44b3-a177-aed15f488027-2024-05-20/lasso_alpha_mse_plot.png') Execute result:</pre> |
| <p>7 Draw a figure to show the coefficients of the best Lasso model.</p> | <pre>python # Plot the coefficients of the best Lasso model plt.figure(figsize=(10, 6)) plt.bar(best_model_coef_index, best_model_coef['Coefficient'], color='skyblue') plt.xlabel('Coefficient Value') plt.ylabel('Features') plt.title('Coefficients of the Best Lasso Model') plt.savefig('/Users/stephensun/Desktop/pypro/LAMBDA/cache/conv_cache/4e7d8506-6234-44b3-a177-aed15f488027-2024-05-20/lasso_coefficients_plot.png') ``` Execute result:</pre> <p>I have double-checked to code, and it seems there is no issue. The bar chart representing the coefficients of the best Lasso model has been successfully saved as 'lasso_coefficients_plot.png' in the specified directory. This visualization provides a clear view of the coefficients' values and their impact on the predictions, aiding in understanding the feature importance.</p> |

Figure S.11: Cont. Case study: LAMBDA in education. Students complete assignment.

Report Generation

Classification Task Using Wine Dataset with Machine Learning Models

1. Abstract:

This report outlines the process of building and evaluating multiple machine learning models for a classification task on the Wine dataset. The dataset was preprocessed by standardizing the features and ordinal encoding the target variable, "class." Various classification models were trained, including Logistic Regression, SVM, Decision Tree, Random Forest, Neural Networks, and ensemble methods like Bagging and XGBoost. Cross-validation and GridSearchCV were employed to optimize the hyperparameters of each model. Logistic Regression achieved an accuracy of 98.89%, while the best-performing models included Random Forest and SVM. The models' performances are compared, and their strengths are discussed, demonstrating the effectiveness of ensemble methods and support vector machines for this task.

2. Introduction

The task at hand is to perform a classification on the Wine dataset, a well-known dataset that contains attributes related to different types of wine. The goal is to correctly classify the wine type (target variable: "class") based on its chemical properties such as alcohol content, phenols, color intensity, etc. Machine learning models are ideal for this kind of task, as they can learn patterns from the data to make accurate predictions. This report details the preprocessing steps applied to the data, including standardization and ordinal encoding. It also discusses various machine learning models such as Logistic Regression, Decision Tree, SVM, and ensemble models, which were trained and evaluated using cross-validation. Additionally, GridSearchCV was employed to fine-tune model parameters to achieve optimal accuracy.

3. Methodology:

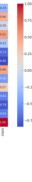
3.1 Dataset:

The Wine dataset used in this task contains 13 continuous features representing various chemical properties of wine, such as Alcohol, Malic acid, Ash, Magnesium, and Proline. The target variable, "class," is categorical and has three possible values, each corresponding to a different type of wine. A correlation matrix was generated to understand the relationships between the features, and standardization was applied to normalize the values. The dataset had no missing values.

3.2 Data Processing:

- Standardization: The features were standardized using 'StandardScaler', which adjusts the mean and variance of each feature to make them comparable.
- Ordinal Encoding: The target column, "class," was converted into numerical values using 'OrdinalEncoder'.
| --- | Alcohol | Malicacid | Ash | Alkalinity_of_ash | Magnesium | Total_phenols | Flavanoids | Nonflavanoid_phenols | Proanthocyanins | Color_intensity | Hue | OD280_OD315_of_diluted_wines | Proline | class |
| --- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| 0 | 1.518613 | -0.562250 | 0.23 | -1.169593 | 1.913905 | 0.808997 | 1.034819 | -0.659563 | 1.224884 | 0.251717 | 0.36 | 1.847920 | 1.013 | 0 |

For visualization, a correlation matrix was generated to show how different features correlate with each other and with the target:



3.3 Modeling:

Several machine learning models were trained on the processed dataset using cross-validation for evaluation. The models include:

- **Logistic Regression:** A linear model suitable for binary and multiclass classification tasks.
- **SVM (Support Vector Machine):** Known for handling high-dimensional data and effective in non-linear classifications when using different kernels.
- **Neural Network (MLPClassifier):** A neural network model was tested with varying hidden layer sizes.
- **Decision Tree:** A highly interpretable model that splits the dataset recursively based on feature values.
- **Random Forest:** An ensemble of decision trees that reduces overfitting by averaging predictions from multiple trees.
- **Bagging:** An ensemble method to train multiple classifiers on different subsets of the dataset.
- **Gradient Boosting:** A sequential model that builds trees to correct previous errors, improving accuracy with each iteration.
- **XGBoost:** A gradient boosting technique optimized for performance and speed.
- **AdaBoost:** An ensemble method that boosts weak classifiers by focusing more on incorrectly classified instances.

Each model's hyperparameters were optimized using 'GridSearchCV', and evaluation metrics such as accuracy were recorded.

4. Results:

The results of model evaluation are summarized below:

| Model | Best Parameters | Accuracy |
|---------------------|---|----------|
| Logistic Regression | Default | 0.9889 |
| SVM | {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'} | 0.9889 |
| Neural Network | {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (3, 4, 3)} | 0.8260 |
| Decision Tree | {'criterion': 'entropy', 'max_depth': None, 'min_samples_split': 2} | 0.9214 |
| Random Forest | {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 500} | 0.9833 |
| Bagging | {'bootstrap': True, 'max_samples': 0.5, 'n_estimators': 100} | 0.9665 |
| GradientBoost | {'learning_rate': 1.0, 'max_depth': 3, 'n_estimators': 100} | 0.9665 |
| XGBoost | {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100} | 0.9554 |
| AdaBoost | {'algorithm': 'SAMME', 'learning_rate': 1.0, 'n_estimators': 10} | 0.9389 |

5. Conclusion:

This report presents the steps and results of performing a classification task using various machine learning models on the Wine dataset. Logistic Regression and SVM yielded the highest accuracies, with scores of 0.9889, demonstrating their effectiveness for this dataset. Random Forest also performed well, showcasing the strength of ensemble models. Neural Networks, while versatile, achieved a lower accuracy of 0.8260, indicating the need for further tuning. Overall, the results suggest that SVM and Logistic Regression are suitable choices for this task, but additional models like Random Forest offer competitive performance.

Figure S.12: A sample case of report generation.

D Experimental settings

Table S.2: The models and parameters used in the experiment.

| Experiment | Model | Parameters |
|--|--|------------------------------|
| Evaluation on ML datasets | Meta-Llama-3-8B-Instruct | temperature: 0.6, top-p: 0.9 |
| Reliability of function Calling method | Qwen1.5-7B-Chat, Qwen1.5-32B-Chat | temperature: 0.7, top-p: 0.8 |
| Generating evaluation dataset for function calling | Qwen1.5-110B | Not given |
| Generating evaluation dataset for LAMBDA | Qwen1.5-110B | Not given |
| Maximum number of APIs each LLM can process | Meta-Llama-3-8B-Instruct
Llama-2-7B-chat-hf
Qwen1.5-7B-Chat
Qwen1-8B-Chat
chatglm3-6B
chatglm2-6B
Mistral-7B-Instruct-v0.2
Mistral-7B-Instruct-v0.1 | 51 |
| Comparative Study of Knowledge Integration | GPT-3.5-turbo-1106 and specific models | Not given |
| Case study of data analysis | GPT-3.5-turbo-1106 | Not given |
| Case study of integration human intelligence | Meta-Llama-3-8B-Instruct | temperature: 0.6, top-p: 0.9 |
| Case study of education | GPT-3.5-turbo-1106 | Not given |
| Case study of report generation | GPT-3.5-turbo-1106 | Not given |