LLM-Based Multi-Agent Blackboard System for Information Discovery in Data Science

Alireza Salemi[1,†] **Mihir Parmar**[2] **Palash Goyal**[2] **Yiwen Song**[2]
Jinsung Yoon[2] **Hamed Zamani**[1] **Tomas Pfister**[2,*] **Hamid Palangi**[2,*]
[1]University of Massachusetts Amherst [2]Google Cloud AI Research
{asalemi,zamani}@cs.umass.edu
{mihirparmar,palashgoyal,yiwensong,jinsungyoon,tpfister,hamidpalangi}@google.com

Abstract

Advances in large language models (LLMs) have created new opportunities in data science, but their deployment is often limited by the challenge of finding relevant data in large data lakes. Existing methods struggle with this: both single- and multi-agent systems are quickly overwhelmed by large, heterogeneous files, and master–slave multi-agent systems rely on a rigid central controller that requires precise knowledge of each sub-agent's capabilities, which is not possible in large-scale settings where the main agent lacks full observability over sub-agents' knowledge and competencies. We propose a novel multi-agent paradigm inspired by the blackboard architecture for traditional AI models. In our framework, a central agent posts requests to a shared blackboard, and autonomous subordinate agents–either responsible for a partition of the data lake or retrieval from web–volunteer to respond based on their capabilities. This design improves scalability and flexibility by removing the need for a central coordinator to know each agent's expertise or internal knowledge. We evaluate the approach on three benchmarks that require data discovery: KramaBench and modified versions of DSBench and DA-Code. Results show that the blackboard architecture substantially outperforms strong baselines, achieving 13%–57% relative improvements in end-to-end success and up to a 9% relative gain in data discovery F1 over the best baseline.

1 Introduction

The recent developments in LLMs have introduced new paradigms for data science, enabling natural language-based approaches to data interpretation, transformation, and analysis [Jing et al., 2025, Huang et al., 2024, Hong et al., 2025, Wang et al., 2025]. Existing work, however, typically assumes an idealized setting in which relevant datasets are already curated and provided to the model—an assumption that diverges substantially from the practical challenges encountered in real-world data science [Lai et al., 2025]. In practice, a substantial fraction of effort is devoted to locating the appropriate data within large and heterogeneous data lakes, often comprising thousands of loosely organized files—a process that constitutes a major bottleneck before any downstream analysis can be performed [Xu et al., 2021]. We argue that this stage of data discovery is both a critical and underexplored challenge for applying LLMs effectively.

[†]Work done during an internship at Google Cloud AI Research.

[*]Joint last authors.

Corresponding authors: Alireza Salemi (asalemi@cs.umass.edu), Mihir Parmar (mihirparmar@google.com), and Hamid Palangi (hamidpalangi@google.com).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

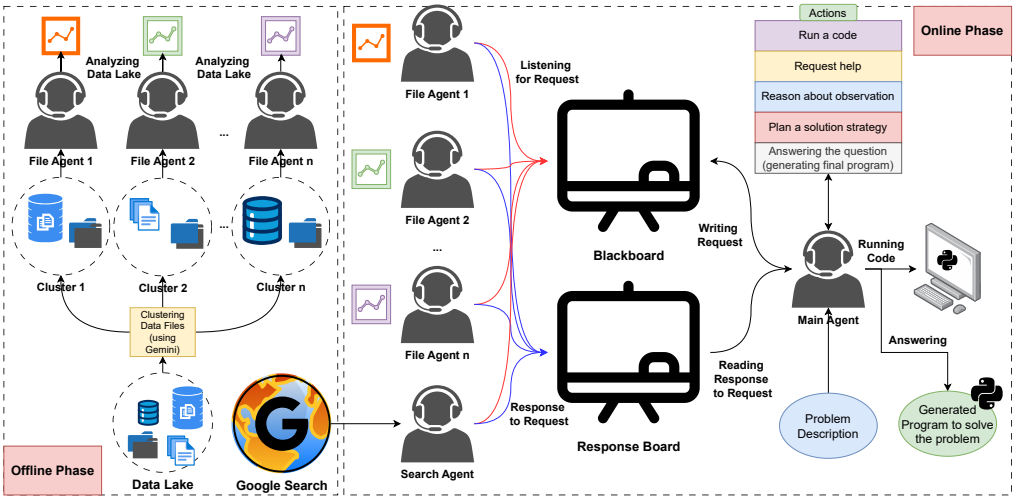


Figure 1: Overview of the blackboard multi-agent system for information discovery in data science. In this framework, the main agent does not assign tasks to subordinate agents. Instead, it posts requests to the blackboard, and subordinate agents autonomously decide whether to respond based on their expertise. The main agent then uses the responses to the request to solve the given task.

Previous work on data science tasks requiring discovery[1] from a data lake has primarily relied on single-agent systems in which an LLM is given access to all files within its context window and is then asked to solve the problem [Lai et al., 2025]. This suffers from several limitations. First, it is not scalable: as the number of files grows, fitting them into the context of an LLM becomes infeasible. Second, the heterogeneity of files poses a challenge, as a single agent may struggle to effectively analyze and integrate diverse forms of information. Third, such systems lack robustness to noise, as the presence of many irrelevant files can overwhelm the model and degrade reasoning and precision. One may argue that Retrieval-Augmented Generation (RAG) [Lewis et al., 2020, Salemi and Zamani, 2024b] provides a solution by choosing a subset of files in the data lake; However, current retrievers are known to perform poorly on tabular and domain-specific data, which are pervasive in data science applications [Yu et al., 2025, Ji et al., 2025, Huang et al., 2022, Gu et al., 2025].

An alternative approach explores multi-agent systems, which frequently adopt a master–slave paradigm [Li et al., 2024, Han et al., 2025, Xu and Peng, 2025]. In this setting, a single controller (e.g., orchestration agent) assigns subtasks to a set of subordinate agents that then execute the specified actions. While conceptually straightforward, this architecture has several drawbacks. First, this master-slave paradigm limits the agents' autonomy: subordinate agents are forced to execute instructions from the coordinator even when they lack sufficient information or hold outdated or erroneous information. Second, the central controller must maintain an accurate model of each agents capabilities to assign tasks, an assumption that is often unrealistic when agents have only partial or evolving knowledge of the problem space. Finally, when multiple agents possess overlapping expertise, the controller faces an inherent assignment ambiguity, making task routing difficult.

Inspired by the blackboard architecture that with substantial impact on traditional AI systems since the 1980s [Erman et al., 1980, Botti et al., 1995], we adopt a new communication paradigm for LLM multi-agent systems: A central agent remains responsible for solving the overall task, similar to the master-slave paradigm; However, rather than assigning subtasks to specific agents, the central agent posts a request on a shared *blackboard* that describes the task or information needed, as shown in Figure 1. Subordinate agents monitoring the blackboard can independently decide whether they possess the capability, knowledge, or interest to contribute to solving process. This design shifts decision-making from a single coordinator to a distributed model whose agents autonomously determine their participation, enabling more flexible collaborations. This differs from the conventional shared-memory paradigm in multi-agent systems. In shared-memory [Sagirova et al., 2025], agents

[1]Examples are shown in Figure 12 and 13 in Appendix E. They require computing/aggregating information from raw data within a data lake, where the specific source files are not pre-identified.

perform assigned tasks based on information in the shared memory, effectively being asked to execute them determined by a central coordinator. Conversely, *in the blackboard architecture, there is no task assignment; instead, requests are broadcast on a blackboard, and each agent retains full autonomy whether to participate in the task or not.*

While the blackboard architecture can be applied broadly within multi-agent systems, its application to data science with data discovery is particularly compelling and underexplored. As shown in Figure 1, the data lake can be partitioned into smaller clusters, e.g., based on similarity, homogeneity, or any criteria facilitating efficient handling, each assigned to a subordinate agent responsible for understanding and processing that subset. The main agent, which is tasked with solving the problem, posts requests on the blackboard specifying the data or general information required. Subordinate agents with the relevant knowledge or capability then autonomously volunteer to respond. This design ensures that each sub-agent manages only a subset of files or web-based information, enhancing scalability compared to approaches that require all data to be loaded into the main agent's prompt. Importantly, the main agent does not need prior knowledge of sub agents knowledge or capability, simplifying coordination and improving flexibility in large-scale data lakes. Here, the main agent's role is primarily to describe the information it requires and define tasks for sub agents, without directly managing or assigning it to them.

We conduct experiments on three datasets for data science requiring an explicit information discovery. KramaBench [Lai et al., 2025] is a recently released benchmark and, to the best of our knowledge, the only publicly available dataset requires explicit data discovery in data science. We also repurpose DSBench [Jing et al., 2025] and DA-Code [Huang et al., 2024] by introducing a data discovery component, making them more challenging than their original formulations. Experimental results demonstrate that the blackboard architecture consistently outperforms strong baselines, including RAG, master-slave multi-agent system, and advanced data science baselines [Hong et al., 2025, Wu et al., 2023], achieving 13% to 57% relative improvement over the best performing baseline in end-to-end problem solving. This improvement is observed across both proprietary and open-source LLMs, highlighting the generalizability of the approach. Furthermore, our method also surpasses baselines in data discovery performance, yielding up to a 9% relative gain in F1 score for correctly identifying relevant files from the data lake. These results underscore the effectiveness of the blackboard architecture as a communication paradigm for multi-agent systems in data science.

2 Problem Formulation

Let $\mathbb{D} = \{d_i\}_{i=1}^{N}$ denote a data lake consisting of N distinct files, each containing information potentially completely or partially relevant to answering a data science question q (some examples of questions are shown in Figures 12 and 13 in Appendix E). The objective is to design a generative system π that, given the query q and the data lake $\mathbb{D}$ as input, produces a program $p \sim \pi(q; \mathbb{D})$ in response. When executed (e.g., using a Python interpreter), this program p retrieves, loads, and processes the appropriate data from $\mathbb{D}$ and solves the problem in the question q to compute the answer. To evaluate the generated program p, we assume the existence of an evaluation function $\mu_{\text{generation}}$ that executes p to produce an output o_p, compares it with the ground-truth response y_q, and assigns a score. In addition, we assume a metric $\mu_{\text{retrieval}}$, given the program p and the ground-truth files $\mathbb{D}_q$, assigns a score reflecting the performance in discovering the correct data sources.

3 Multi-Agent Blackboard System

This section introduces an alternative paradigm for LLM-based multi-agent systems inspired by blackboard systems [Erman et al., 1980], distinct from the widely used master–slave architecture. As outlined in §1, blackboard multi-agent systems provide several advantages over the master-slave approach. Rather than directly assigning tasks to sub-agents, the main agent posts its requests (i.e., sub-tasks for which it requires assistance) on a blackboard as a broadcast channel accessible to all agents. Each helper agent independently evaluates whether it can respond to a request, considering its own capabilities, availability, and other factors. If an agent decides to contribute, it writes its response to the corresponding request, and the main agent then decides whether to use or ignore the provided information. *This way, all agents retain full autonomy over their actions, and no centralized controller forces them to execute a task.* While the blackboard paradigm is applicable to a wide range

of tasks, we focus on data science tasks that require data discovery, where its characteristics are particularly advantageous, as discussed in §1. The remainder of this section explains our method.

Overview: An overview of our method is presented in Figure 1. The system π operates over a data lake $\mathbb{D}$ by first partitioning $\mathbb{D}$ into C clusters. Each cluster $\mathbb{D}_i$ is assigned to a file agent π_{f_i}, which is responsible for handling, loading, processing, and retrieving information from files within its cluster. In addition, a search agent π_s is included to retrieve external information from the web that may be required to solve the problem. The system π is composed of a main agent π_m, responsible for solving the query q, and a set of $C+1$ helper agents $\Pi_{\text{helper}} = \{\pi_{f_i}\}_{i=1}^{C} \cup \{\pi_s\}$ providing specialized assistance. The query q is presented to π_m, which iteratively selects an action $a \in \mathbb{A}$ from the action space $\mathbb{A}$, executes it, and observes the outcomes. Among its actions, the main agent may interact with a blackboard β, a shared communication medium where it can post a request r without addressing a specific sub-agent. The helper agents Π_{helper} continuously monitor the blackboard, determine whether they can address a request, and, if so, provide their outputs on the corresponding response board β_r. These responses are then collected and given to π_m, which incorporates them into its decision-making process.[2] The main agent is limited to at most T sequential actions (including blackboard interactions) to solve the query q, ultimately producing a python program p that answers q.

Clustering Data Lake: There are different methods for partitioning the data lake into clusters; applying clustering algorithms over file representations, random partitioning, or other heuristic methods. For simplicity, by default, we do not utilize file content and instead rely solely on file names. Specifically, the file names are provided to an LLM—Gemini-2.5-Pro—which using the prompt shown in Figure 4, clusters the files into categories based only on their names. An example of this clustering is provided in Figure 11 in Appendix E, where the model successfully groups related files together. For instance, it clusters all files originating from the National Interagency Fire Center into a category labeled "NIFC Wildfire Statistics." The number of automatically derived clusters for each dataset is reported in Table 3 in Appendix A. To further demonstrate generalizability of our method to clustering approach (Appendix F), we conduct an experiment that uses file content embedding with E5-large Wang et al. [2022] with KMeans as an alternative clustering method to our default mode. The details of this experiment are described in Appendix F.

3.1 Main Agent

The primary role of the main agent is to solve the problem in collaboration with sub-agents. The main agent follows the ReAct framework [Yao et al., 2023], where at each step t, given the query q and the history of actions and observations $\mathbb{H}_{t-1}$, it first reasons about what is the best next action and selects one from a predefined action space, executes it, observes the outcome, and appends the resulting observation to update the history $\mathbb{H}_t$.[3] The prompt for the main agent is in Figure 5 in Appendix C. The agent selects one of the following predefined actions in each step:

- ***Planning:*** The LLM decomposes the problem into sub-problems and outlines a plan for addressing them. This action has no effect on the environment; it serves as an internal thinking for the LLM's problem-solving. In response, the system acknowledges the plan and asks the LLM to proceed.
- ***Reasoning:*** The LLM focuses on a specific aspect of the problem and explains its reasoning, analysis, or interpretation of the available observations and steps taken so far in this process. Similar to the planning step, this action has no external effect on the environment but functions as an internal mechanism to guide the LLM's problem-solving process. In response, the system simply acknowledges the reasoning and prompts the LLM to continue.
- ***Executing Code:*** The agent generates python code, which is executed using a python interpreter. If the code runs successfully, the resulting outputs are returned to the agent for observation; otherwise, the agent receives the corresponding error messages. This action enables the agent to explore the problem interactively, inspect data files, and experiment with them to gain a deeper understanding of their content and structure and how to process them.
- ***Requesting Help:*** The agent formulates a request for assistance from the sub-agents, specifying, for example, the types of data files or information needed, or the resources required to apply a tool

[2] Responses aren't written to board β to prevent cross-influence of sub-agents; instead, they are written to a board β_r, enabling independent operation and exclusive access by the main agent.

[3] In this work, model inputs, outputs, and observations are appended to the LLM prompt using its chat-based format.

or solve a sub-problem. This request is posted on the blackboard β for helper agents. Once they respond, if they respond, their responses on the response board β_r are provided back to the main agent as the outcome of this action for observation and further use in its decision-making process.

- ***Answering:*** The agent concludes the problem-solving process by generating a final program that produces the answer to the query. This action terminates the process, and the output of this step constitutes the final program p generated by the system to address the problem.

3.2 Helper Agents

In a data science, information discovery can typically be categorized into two tasks: (1) identifying the specific files that contain the data necessary to the problem, and (2) retrieving general knowledge about concepts relevant to the problem, such as domain-specific terms or details of particular algorithms and methods. To support these, our framework employs two types of helper agents:

File Agent: Handling all files in a data lake with a single agent is infeasible: it involves a large number of files, many of which are lengthy and may exceed the agents context window; the files span diverse topics, which can confuse the agent and hinder effective reasoning; and accessing and processing all files simultaneously can be computationally expensive, leading to unnecessary overhead and slower problem-solving. Thus, in our framework, each file agent is assigned responsibility for a subset of files, as described earlier in the clustering procedure. In an offline phase, the file agent π_{f_i} takes as input a subset of the data lake $\mathbb{D}_i$ and operates through a two-step procedure. First, the agent selects a subset[4] (or all) of the files to examine their content. The contents of them are presented to the agent for inspection (details of presentation are in Appendix D). Secondly, after observing the selected files, the agent analyzes them, learning how they are structured, what pre-processing may be required, and how they should be processed in general. An example of such an analysis is in Figure 10 in Appendix E. Then, in the online phase, the agent listens for requests from the main agent. Upon receiving a request, based on the analysis it did earlier, it determines whether it can contribute to answering it. If so, the agent generates a detailed plan specifying which files in $\mathbb{D}_i$ are relevant, how they should be loaded, what libraries to use, the steps required for data processing, and samples from the data. The prompt used to guide the file agent is shown in Figure 6 in Appendix C.

Search Agent: Certain data science problems require task-specific knowledge about algorithms or domain expertise that the LLM may not possess. Thus, we design a search agent that retrieves information from a search engine. This agent operates according to the prompt shown in Figure 7 in Appendix C. Given a request r on the blackboard β, the agent first determines whether it is capable of addressing the request. It is specifically restricted to general web-based information retrieval and does not respond to requests involving access to local files. If the agent determines feasibility, it enters an iterative search process with a maximum of $T_{\text{search}} = 3$ steps. At each step t, the agent generates a set of queries $\mathbb{Q}_t$, which are submitted to a search engine—Google Custom Search Engine[5]—to retrieve $k = 3$ webpage per query. The content of the webpages are then extracted using *beautifulsoup* library[6] to be presented to the search agent. The extracted documents are then evaluated by the agent to determine whether they provide sufficient information to answer the request. If so, the agent generates a response to the request, which is posted to the response board β_r. If the information is insufficient, a new set of queries is generated to continue gathering relevant data from the web.

4 Experiments

4.1 Experimental Setup

Benchmarks: To the best of our knowledge, KramaBench is the only public benchmark for data science problems that explicitly incorporates a data discovery phase, which we adopt in our evaluation. In addition, we repurpose two existing datasets, DSBench (data analysis task) [Jing et al., 2025] and DA-Code [Huang et al., 2024], to include in this phase. Specifically, we manually filtered out all questions that do not require any data file for answering, as well as those that lack sufficient hints

[4]When filenames indicate multiple files of the same data type over different time periods, the agent can infer the structure from a small representative sample rather than inspecting all files.

[5]We use Google Custom Search Engine, configured to exclude all websites related to datasets to prevent data leakage.

[6]`https://pypi.org/project/beautifulsoup4/`

for data discovery.[7] After filtering, we aggregated all remaining files across questions into a unified data lake, such that the model must perform discovery to identify relevant files at inference time. In this setup, only the question and the data lake are provided to the model, requiring it to identify the relevant files to answer the question, following the same protocol as KramaBench. Further details on this filtering process, along with dataset statistics in Table 3, are provided in Appendix A.

Evaluation: We execute each generated program and compare its output against the ground-truth for the corresponding question. For each dataset, we adopt its evaluation protocol. For KramaBench, we use the official evaluation script in its repository.[8] For DA-Code, we rely on the official evaluation script by authors.[9] For DSBench, we use the original evaluation, in which an LLM serves as the judge. The generated programs output is compared against the reference using Gemini-2.5-Pro as the judge LLM, with the evaluation prompt shown in Figure 3 in Appendix A, producing a binary score.

Inference Setup: We set the maximum actions of the main agent to $T = 10$. We use nucleus sampling [Holtzman et al., 2020] with a temperature of 0.1 for more deterministic inference and default value for other hyperparameters. Proprietary models are accessed via Vertex AI, while open-source models are served by vLLM Kwon et al. [2023]. At each step, we cap the number of generated tokens at 8,192. We use Gemini-2.5-Pro and -Flash [Gemini-Team, 2025], and Claude-4-Opus [Anthropic, 2025] as the proprietary and Qwen3-Coder[10] with 30 billion parameters [Qwen-Team, 2025] as the open-source LLMs. Experiments are conducted on 2 NVIDIA A100 (80GB) GPUs.

Baselines: To evaluate our method against alternative approaches for solving data science problems involving data discovery, we compare it with the following baselines:

- ***DS-GRU:*** We adopt the only existing baseline (to the best of our knowledge) for data discovery in data science problems, which appends all available files directly into the LLM prompt and attempts to solve the problem [Lai et al., 2025]. This baseline uses a self-correction loop that retries when errors occur in generated codes. For details, we refer the reader to Lai et al. [2025].
- ***Retrieval-Augmented Generation (RAG):*** This retrieves the top 5 files[11] based on the file names and contents (presenting file content to the LLM is detailed in Appendix D) from the data lake using E5-large[12] [Wang et al., 2022], a 330M-parameter embedding model and use it to solve the problem. It then follows the same ReAct procedure as the main agent in § 3.1, with two key modification: 1) the retrieved files contents are presented directly to the LLM's prompt and 2) the help-request action is replaced with a restricted action that permits requests only to the search agent. This isolates the effect of replacing the file discovery mechanism with RAG, allowing a controlled study of its impact on performance. The prompt used is shown in Figure 9 in Appendix C.
- ***Master-Slave:*** This follows the same procedure as the main agent described in §3.1. The key difference is that, instead of requests on the blackboard, the agent directly invokes sub-agents (consisting of the search agent and the file agents as explained in §3.2) by referencing their names and assign task to them. The prompt used for this baseline is shown in Figure 8 in Appendix C.

4.2 Empirical Findings

Main Results: We conduct experiments on the datasets described in §4.1. The results are presented in Table 1. These results demonstrate that our method, the Blackboard System, outperforms all baselines on average across all the datasets. Specifically, the Blackboard System surpasses the DS-GRU, RAG and Master-Slave approaches on all three datasets and achieves similar or higher performance in 4 out of 6 categories on KramaBench. Furthermore, we observe that the Blackboard System consistently outperforms the baselines regardless of the backbone LLM, highlighting its robustness and generalizability. This advantage also holds with the more recent Gemini 3.1 Pro, where Blackboard outperforms Master–Slave on all three benchmarks (37.79% vs. 31.80% on average; Table 6 in Appendix F). We attribute this improvement to the design of the Blackboard System,

[7] E.g., questions that ask for a data science metric on a column w/o specifying the structure or content of the relevant file.
[8] `https://github.com/mitdbg/KramaBench`
[9] `https://github.com/yiyihum/da-code`
[10] `https://huggingface.co/Qwen/Qwen3-Coder-30B-A3B-Instruct`
[11] This is chosen based on the average files needed to solve problems (1.6) and the context size of backbone LLMs used.
[12] `https://hf.co/intfloat/e5-large-v2`

Table 1: Results on the benchmarks. The best results for each LLM are highlighted in **bold**. The KramaBench categories are abbreviated: Arc. (Archaeology), Ast. (Astronomy), Bio. (Biomedical), Env. (Environment), Leg. (Legal), and Wild. (Wildfire).

Method	LLM	KramaBench							DS Bench	DA-Code	Avg.
		Arc.	Ast.	Bio.	Env.	Leg.	Wild.	Avg.			
DS-GRU	Qwen3-Coder	**0.00%**	1.80%	2.11%	1.15%	3.27%	13.54%	3.64%	0.00%	0.00%	1.21%
RAG		**0.00%**	3.16%	4.99%	0.54%	6.19%	16.93%	5.30%	6.32%	0.00%	3.87%
Master-Slave		**0.00%**	3.55%	3.39%	**7.77%**	**8.90%**	21.79%	7.56%	7.55%	0.00%	5.03%
Blackboard		**0.00%**	**7.69%**	**7.85%**	4.47%	6.36%	**23.97%**	**8.39%**	**14.22%**	**1.11%**	**7.90%**
DS-GRU	Gemini 2.5 Flash	0.00%	**7.83%**	0.09%	10.93%	12.46%	13.34%	7.44%	5.53%	0.00%	4.32%
RAG		**16.66%**	3.57%	13.98%	**28.57%**	10.97%	33.67%	17.90%	22.92%	**2.75%**	14.52%
Master-Slave		**16.66%**	3.16%	13.98%	17.46%	21.75%	25.80%	16.46%	26.48%	0.55%	14.49%
Blackboard		**16.66%**	3.57%	**14.78%**	22.92%	**27.09%**	**41.04%**	**21.01%**	**28.06%**	0.55%	**16.54%**
DS-GRU	Gemini 2.5 Pro	25.00%	6.69%	10.64%	27.47%	5.94%	39.36%	19.18%	3.95%	0.00%	7.71%
RAG		**33.33%**	8.47%	32.53%	31.36%	25.55%	38.32%	28.26%	27.27%	0.00%	18.51%
Master-Slave		**33.33%**	8.47%	24.74%	32.81%	34.64%	58.98%	32.16%	34.38%	5.49%	24.01%
Blackboard		**33.33%**	**17.95%**	**36.83%**	**39.31%**	**34.92%**	**62.88%**	**37.53%**	**38.73%**	**9.34%**	**28.53%**
DS-GRU	Claude 4 Opus	8.33%	1.38%	1.90%	8.14%	9.80%	23.14%	8.78%	3.55%	0.00%	4.11%
RAG		**33.33%**	11.52%	23.42%	31.61%	31.80%	45.80%	29.58%	35.57%	3.85%	23.00%
Master-Slave		**33.33%**	8.69%	32.28%	**39.16%**	**44.08%**	48.35%	34.31%	45.84%	2.75%	27.63%
Blackboard		**33.33%**	**18.69%**	**45.31%**	34.35%	42.48%	**50.06%**	**37.37%**	**49.80%**	**7.14%**	**31.43%**

where tasks are not explicitly assigned to helper agents; instead, each agent autonomously decides whether to participate based on its capabilities. This self-selection enhances both problem-solving efficiency and data discovery performance. We further compare the Blackboard system with two advanced methods for data analysis, Data Interpreter [Hong et al., 2025] and AutoGen [Wu et al., 2023]. Figure 2(A) shows average performance on KramaBench, with per-task results in Table 5 (Appendix F). The Blackboard system outperforms these baselines on five of six tasks and on average, indicating more effective adaptation and coordination across heterogeneous data science workloads.

File Discovery Performance: To analyze the effectiveness of different methods in data discovery, we report recall, precision, and F1-score for the file discovery task, i.e., identifying the correct files required to answer each question. The results of this experiment, using Gemini 2.5 Pro as the LLM, are presented in Table 2. The results indicate that the blackboard system achieves the highest recall, precision, and F1-score, both on average and across the three datasets. In particular, for KramaBench, the blackboard system attains the highest F1-score in 4 out of 6 domains. We attribute this improvement to the design of our method, where the main agent does not directly assign requests to specific file agents, as in the master–slave setup. Instead, each file agent independently decides whether it can contribute based on its data holdings, leading to more accurate file discovery.

Effect of Number of Main Agent's Actions on Performance: To evaluate the effect of the main agent's maximum action limit, we vary it across $2, 4, 6, 8, 10, 15$ and evaluate the blackboard system on KramaBench using Gemini 2.5 Pro as the backbone LLM. As shown in Figure 2(B) (average performance) and Figure 19 in Appendix F (per-task results), increasing the action budget leads to consistent performance gains up to 10 steps, after which improvements plateau. This suggests that a larger budget enables the agent to more thoroughly explore the solution space, evaluate alternative strategies, and produce higher-quality outputs, with diminishing returns beyond this point.

Effect of Search Agent on Performance: We observed the LLM sometimes lacks the domain-specific knowledge or familiarity with specialized algorithms needed to fully solve problems. To address this, we include a search agent that retrieves relevant external information. Comparing the blackboard system with and without the search agent on KramaBench (Figure 2(C) for average and Figure 18 in Appendix F for each task, using Gemini 2.5 Pro) shows that incorporating the search agent improves average performance. Analysis reveals that when the main agent encounters unfamiliar concepts, it requests information from the web, which the search agent retrieves, allowing problem-solving to continue effectively. Examples illustrating this behavior are in Figures 12 and 13 in Appendix E, highlighting the search agent's role when external domain knowledge is essential.

Table 2: File discovery performance (Gemini-2.5-Pro). The best results are highlighted in **bold**. The KramaBench categories are abbreviated: Arc. (Archaeology), Ast. (Astronomy), Bio. (Biomedical), Env. (Environment), Leg. (Legal), and Wild. (Wildfire).

Method	Metric	KramaBench							DS-Bench	DA-Code	Avg.
		Arc.	Ast.	Bio.	Env.	Leg.	Wild.	Avg.			
RAG	Recall	0.875	0.125	**0.666**	0.350	0.127	0.238	0.396	0.035	0.257	0.229
	Precision	**1.000**	0.125	0.666	0.450	0.133	0.452	0.471	0.047	0.456	0.324
	F1	0.916	0.125	0.629	0.332	0.105	0.301	0.401	0.034	0.307	0.247
Master-Slave	Recall	**0.916**	0.5138	0.648	0.382	**0.444**	**0.567**	0.578	0.323	0.546	0.482
	Precision	0.930	**0.750**	**0.722**	0.500	**0.494**	**0.642**	0.673	0.503	0.767	0.647
	F1	0.913	0.577	**0.674**	0.389	**0.450**	**0.576**	0.596	0.358	0.584	0.513
Black-board	Recall	**0.916**	**0.576**	0.648	**0.604**	0.383	0.464	**0.598**	**0.402**	**0.600**	**0.533**
	Precision	**1.000**	0.733	**0.722**	**0.703**	0.302	0.603	**0.677**	**0.584**	**0.837**	**0.699**
	F1	**0.944**	**0.618**	**0.674**	**0.588**	0.304	0.495	**0.603**	**0.438**	**0.643**	**0.561**

Scalability Comparison between Blackboard and Master–Slave Systems: We report relative performance gains rather than absolute score differences to enable a fair comparison across datasets with varying difficulty and score ranges. Absolute improvements are not directly comparable across tasks because they depend on baseline accuracy and the attainable upper bound; for instance, a 5-point increase is minor when the baseline is already high but more meaningful for a challenging task with low initial performance. In contrast, relative gain measures the proportional improvement with respect to the baseline and therefore provides a normalized metric across heterogeneous tasks. Using this normalized measure, we analyze how the Blackboard architecture scales relative to the Master–Slave baseline as a function of data lake size (Figure 20(A) in Appendix F). Each point in the figure corresponds to a task domain, and the regression line shows a positive correlation between data lake size and relative performance gain. Consistent with this trend, Blackboard achieves its largest gains on the two datasets with the largest data lakes, DA-Code (145 files) and Astronomy from KramaBench (1556 files), with relative improvements of 70% and 112%, respectively.

To isolate the effect of data lake size from domain and task difficulty, we further keep the tasks and ground-truth files of these two datasets fixed and vary only the number of distractor files: as the data lake grows from 33% to 67% and 100% of its full size, the relative gain increases from 15% to 49% and 70% on DA-Code, and from 34% to 74% and 112% on Astronomy (Figure 20(B)). Together, these results show that the Blackboard architecture scales more effectively in large data lakes, while the Master–Slave provides limited gains as the data lake grows.

Runtime and Cost Analysis: To characterize the efficiency–cost trade-off of the Blackboard system relative to the RAG and Master–Slave baselines, we randomly sample 50 questions from the KramaBench benchmark spanning all 6 different domains in the benchmark and measure both runtime and cost per question for each method. Unlike RAG and Master–Slave, which execute their component calls sequentially following the ReAct framework, the Blackboard architecture parallelizes sub-agent interactions: once the main agent posts a request to the shared blackboard, the corresponding sub-agents process it independently. As shown in Figure 2(D, E) with more details in Figure 21 in Appendix F, the runtime of all three systems lies in a narrow range (132.0–145.2 seconds), indicating no substantial difference in latency. In terms of monetary cost, Blackboard is more expensive per question (approximately $2.3\times$ the cost of RAG and $1.8\times$ that of Master–Slave), due to increased token usage. However, this additional cost yields substantial performance gains—54.1% over RAG and 18.8% over Master–Slave—so that Blackboard achieves higher answer quality while maintaining comparable runtime, resulting in a favorable accuracy–cost trade-off.

To examine whether these gains can be attributed solely to the higher inference cost, we further conduct a budget-matched experiment in which the main agent of the Master–Slave system continues beyond its original limit of $T = 10$ actions until its cost per question matches that of Blackboard. Even with this additional budget, Blackboard remains better on all three benchmarks (37.53% vs. 33.23% on KramaBench, 38.73% vs. 35.57% on DSBench, and 9.34% vs. 7.14% on DA-Code; Table 4 in Appendix F), indicating that its gains stem from how this budget is used—broadcasting requests and allowing agents to self-select—rather than from the additional computation alone.



Figure 2: Performance and Efficiency Evaluation of the Blackboard System. **(A)** Performance vs. advanced baselines; **(B)** performance scalability with max actions; **(C)** impact of search component; **(D)** cost analysis per question; **(E)** runtime comparison showing efficiency gains; **(F)** content-based vs. filename clustering; and **(G)** optimization of cluster size k.

Effect of Clustering Strategy and Number of Clusters on Performance: As described in §3, we use Gemini 2.5 Pro to cluster files based on filenames, since providing full file contents is infeasible. To study content-based clustering, we instead encode file contents using the E5-Large embedding model and apply KMeans [Lloyd, 1982]. Because KMeans requires a predefined number of clusters, we consider 2, 4, 8, and 16 clusters and restrict our analysis to datasets with at least 100 files (the Legal and Astronomy subsets of KramaBench and DA-Code). Results in Figure 2(G) (average performance) and Figure 22 in Appendix F (per dataset) show that increasing the number of clusters generally improves performance, as each sub-agent operates on a smaller, more coherent subset of files. Figure 2(F) (average) and Figure 23 in Appendix F (per task) further compare filename-based clustering with Gemini to content-based clustering, showing that semantic content clustering outperforms the alternatives. The results show that our method supports embedding-based clustering and is not tied to a specific clustering approach.

Case Studies: To qualitatively analyze the blackboard system, we present several case studies:

- **Requests on the blackboard:** An example request on the blackboard is shown in Figure 14 in Appendix E. Given a question, the main agent formulates a request specifying likely column names, data formats, and interpretation guidance. In response, several helper agents (3 of 8 in this case) contribute. Although the relevant files are distributed across different clusters, each responding agent independently provides file locations, data-loading code, and descriptions of the data structure with suggested processing steps. These responses cover all ground-truth files needed to answer the question, showing how the blackboard mechanism enables information discovery and integration.
- **Comparing Generated Program by Blackboard with Master-Slave System:** To further illustrate this difference, Figure 15 in Appendix E compares programs generated by the Blackboard and Master–Slave systems. The Blackboard agent produces a better solution by correctly interpreting the prompt and selecting the appropriate data files: it identifies the patients' `Age` in `mmc1.xlsx` and the requested `APP-Z score` in `mmc7.xlsx`. In contrast, the Master–Slave agent misinterprets the request and uses a general protein abundance score (`APP_log2_abundance`) from `mmc2.xlsx`. This selection error leads to an incorrect result of `74`, whereas the Blackboard's accurate data discovery and reasoning yield the correct answer of `60`.

5 Related Work

LLMs for Data Science: Specialized benchmarks evaluate LLMs in data science. DS-1000 [Lai et al., 2023], ARCADE [Yin et al., 2023], DataSciBench [Zhang et al., 2025], and DSEval [Zhang et al., 2024] focus on translating natural language instructions into correct implementations, unlike broader programming benchmarks such as SWE-Bench [Jimenez et al., 2024], ML-Bench [Tang et al., 2025], and BigCodeBench [Zhuo et al., 2025]. Most assume relevant data files are pre-specified, though DSBench [Jing et al., 2025] and BLADE [Gu et al., 2024] address multi-step reasoning, while ScienceAgentBench [Chen et al., 2025] and BixBench [Mitchener et al., 2025] focus on domain knowledge integration. These benchmarks largely overlook the challenge of discovering relevant data in large, heterogeneous repositories—a gap addressed by KramaBench [Lai et al., 2025], which explicitly evaluates data discovery. Building on this, we study how agents can autonomously identify and leverage correct data sources for end-to-end analysis. Applications of LLMs in data science have evolved from single-turn code generation to interactive, tool-augmented agents using models specialized for code, including GPT [Brown et al., 2020], CodeGen [Rubavicius et al., 2025], StarCoder [Li et al., 2023], and Code Llama [Rozière et al., 2024]. While few-shot prompting [Brown et al., 2020] remains effective, state-of-the-art methods adopt agentic or multi-agentic frameworks combining reasoning with tool use. ReAct [Yao et al., 2023] interleaves reasoning with actions, extended to execution environments [Chen et al., 2019]. Toolformer [Schick et al., 2023] and Gorilla [Patil et al., 2024] train LLMs to call APIs for tasks requiring specialized libraries. Self-correction frameworks like Self-Debug [Chen et al., 2024] and Reflexion [Shinn et al., 2023] refine code using execution feedback. To improve reliability, many systems integrate RAG [Lewis et al., 2020, Salemi et al., 2025, Salemi and Zamani, 2025, 2024a] to access documentation or code examples, reducing hallucinations and ensuring up-to-date library use. Multi-agent master–slave frameworks, such as AutoKaggle [Li et al., 2024], further demonstrate promising results in this domain. A more detailed related work regarding this is discussed in Appendix B.

Blackboard Systems: The blackboard system is a architectural model from classical AI, developed for problems that require incremental and opportunistic reasoning. It was implemented in the Hearsay-II speech understanding [Erman et al., 1980] and is characterized by three components: (1) a global, hierarchical data structure (the blackboard) that maintains the current state of the solution; (2) independent specialist modules, known as knowledge sources, which monitor the blackboard and contribute partial solutions; and (3) a control mechanism that opportunistically determines which knowledge source to activate next [Nii, 1986]. Following successful applications in domains such as sonar interpretation with the HASP/SIAP system [Nii et al., 1982], the architecture evolved to incorporate more sophisticated control strategies. Inspired by this, we adapt the blackboard architecture for LLM-based multi-agent communication. Instead of a central controller assigning tasks, agents operate autonomously and respond to requests posted on the blackboard when they can contribute without being forced to participate. The main agent then integrates the information provided by the sub-agents to solve the problem. Detailed related work is discussed in Appendix B.

6 Conclusions & Future Work

We addressed the critical challenge of data discovery in large heterogeneous data lakes, a key bottleneck for applying LLMs in data science. We introduced a novel multi-agent paradigm based on the blackboard architecture, which replaces rigid centralized task assignment with a flexible, decentralized model of agent collaboration. Extensive experiments on three data science benchmarks show that our framework consistently outperforms strong baselines, including RAG and the master–slave paradigm, achieving up to 57% relative improvement in end-to-end task success. These results highlight the importance of communication architecture in multi-agent systems and establish the blackboard paradigm as a scalable, flexible, and effective solution for complex data science workflows. Future work could extend the blackboard architecture beyond data science, as the proposed approach is general and applicable to a wide range of multi-agent systems. Another promising direction is to investigate more adaptive strategies for data partitioning among agents, enabling the system to better handle dynamic and evolving data environments.

Acknowledgments and Disclosure of Funding

This work was supported in part by the Center for Intelligent Information Retrieval, in part by NSF grant number 2402873, in part by the Office of Naval Research contract number N000142412612, and with support from Google.org. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

References

Anthropic. System card: Claude opus 4 & claude sonnet 4. https://www-cdn.anthropic.com/4263b940cabb546aa0e3283f35b686f4f3b2ff47.pdf, May 2025. Also referenced in the official release blog post: https://www.anthropic.com/news/claude-4.

V. Botti, F. Barber, A. Crespo, E. Onaindia, A. Garcia-Fornes, I. Ripoll, D. Gallardo, and L. Hernández. A temporal blackboard for a multi-agent environment. *Data & Knowledge Engineering*, 15(3): 189–211, 1995. ISSN 0169-023X. doi: https://doi.org/10.1016/0169-023X(95)00007-F. URL https://www.sciencedirect.com/science/article/pii/0169023X9500007F.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

X. Chen, C. Liu, and D. Song. Execution-guided neural program synthesis. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1gfOiAqYm.

X. Chen, M. Lin, N. Schärli, and D. Zhou. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KuPixIqPiq.

Z. Chen, S. Chen, Y. Ning, Q. Zhang, B. Wang, B. Yu, Y. Li, Z. Liao, C. Wei, Z. Lu, V. Dey, M. Xue, F. N. Baker, B. Burns, D. Adu-Ampratwum, X. Huang, X. Ning, S. Gao, Y. Su, and H. Sun. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6z4YKr0GK6.

L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Comput. Surv.*, 12(2):213–253, June 1980. ISSN 0360-0300. doi: 10.1145/356810.356816. URL https://doi.org/10.1145/356810.356816.

Gemini-Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025. URL https://arxiv.org/abs/2507.06261.

K. Gu, R. Shang, R. Jiang, K. Kuang, R.-J. Lin, D. Lyu, Y. Mao, Y. Pan, T. Wu, J. Yu, Y. Zhang, T. M. Zhang, L. Zhu, M. A. Merrill, J. Heer, and T. Althoff. BLADE: Benchmarking language model agents for data-driven science. In Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 13936–13971, Miami, Florida, USA, Nov. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.815. URL https://aclanthology.org/2024.findings-emnlp.815/.

K. Gu, Z. Zhang, K. Lin, Y. Zhang, A. Paruchuri, H. Yu, M. Kazemi, K. Ayush, A. A. Heydari, M. A. Xu, G. Narayanswamy, Y. Liu, M.-Z. Poh, Y. Yang, M. Malhotra, S. Patel, H. Palangi, X. Xu, D. McDuff, T. Althoff, and X. Liu. Radar: Benchmarking language models on imperfect tabular data, 2025. URL https://arxiv.org/abs/2506.08249.

B. Han and S. Zhang. Exploring advanced llm multi-agent systems based on blackboard architecture, 2025. URL https://arxiv.org/abs/2507.01701.

S. Han, Q. Zhang, Y. Yao, W. Jin, and Z. Xu. Llm multi-agent systems: Challenges and open problems, 2025. URL https://arxiv.org/abs/2402.03578.

A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

S. Hong, Y. Lin, B. Liu, B. Liu, B. Wu, C. Zhang, D. Li, J. Chen, J. Zhang, J. Wang, L. Zhang, L. Zhang, M. Yang, M. Zhuge, T. Guo, T. Zhou, W. Tao, R. Tang, X. Lu, X. Zheng, X. Liang, Y. Fei, Y. Cheng, Y. Ni, Z. Gou, Z. Xu, Y. Luo, and C. Wu. Data interpreter: An LLM agent for data science. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19796–19821, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1016. URL https://aclanthology.org/2025.findings-acl.1016/.

J. Huang, W. Zhong, Q. Liu, M. Gong, D. Jiang, and N. Duan. Mixed-modality representation learning and pre-training for joint table-and-text retrieval in OpenQA. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 4117–4129, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-emnlp.303. URL https://aclanthology.org/2022.findings-emnlp.303/.

Y. Huang, J. Luo, Y. Yu, Y. Zhang, F. Lei, Y. Wei, S. He, L. Huang, X. Liu, J. Zhao, and K. Liu. DA-code: Agent data science code generation benchmark for large language models. In Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13487–13521, Miami, Florida, USA, Nov. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.748. URL https://aclanthology.org/2024.emnlp-main.748/.

X. Ji, P. Glenn, A. G. Parameswaran, and M. Hulsebos. Target: Benchmarking table retrieval for generative tasks, 2025. URL https://arxiv.org/abs/2505.11545.

C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. R. Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

L. Jing, Z. Huang, X. Wang, W. Yao, W. Yu, K. Ma, H. Zhang, X. Du, and D. Yu. DSBench: How far are data science agents from becoming data science experts? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=DSsSPr0RZJ.

W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica. 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, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

E. Lai, G. Vitagliano, Z. Zhang, S. Sudhir, O. Chabra, A. Zeng, A. A. Zabreyko, C. Li, F. Kossmann, J. Ding, J. Chen, M. Markakis, M. Russo, W. Wang, Z. Wu, M. J. Cafarella, L. Cao, S. Madden, and T. Kraska. Kramabench: A benchmark for ai systems on data-to-insight pipelines over data lakes, 2025. URL https://arxiv.org/abs/2506.06541.

Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W.-t. Yih, D. Fried, S. Wang, and T. Yu. Ds-1000: a natural and reliable benchmark for data science code generation. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

R. Li, L. B. allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. LI, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, J. Lamy-Poirier, J. Monteiro, N. Gontier, M.-H. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. Murthy, J. T. Stillerman, S. S. Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, U. Bhattacharyya, W. Yu, S. Luccioni, P. Villegas, F. Zhdanov, T. Lee, N. Timor, J. Ding, C. S. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. Hughes, T. Wolf, A. Guha, L. V. Werra, and H. de Vries. Starcoder: may the source be with you! *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=KoFOg41haE. Reproducibility Certification.

Z. Li, Q. Zang, D. Ma, J. Guo, T. Zheng, M. Liu, X. Niu, Y. Wang, J. Yang, J. Liu, W. Zhong, W. Zhou, W. Huang, and G. Zhang. Autokaggle: A multi-agent framework for autonomous data science competitions, 2024. URL https://arxiv.org/abs/2410.20424.

S. P. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inf. Theory*, 28:129–136, 1982. URL https://api.semanticscholar.org/CorpusID:10833328.

L. Mitchener, J. M. Laurent, B. Tenmann, S. Narayanan, G. P. Wellawatte, A. White, L. Sani, and S. G. Rodriques. Bixbench: a comprehensive benchmark for llm-based agents in computational biology, 2025. URL https://arxiv.org/abs/2503.00096.

H. P. Nii. The blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*, 7(2):38, Jun. 1986. doi: 10.1609/aimag.v7i2.537. URL https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/537.

H. P. Nii, E. A. Feigenbaum, and J. J. Anton. Signal-to-symbol transformation: Hasp/siap case study. *AI Magazine*, 3(2):23, Jun. 1982. doi: 10.1609/aimag.v3i2.368. URL https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/368.

S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=tBRNC6YemY.

Qwen-Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve. Code llama: Open foundation models for code, 2024. URL https://arxiv.org/abs/2308.12950.

R. Rubavicius, A. V. Miceli-Barone, A. Lascarides, and S. Ramamoorthy. Conversational code generation: a case study of designing a dialogue system for generating driving scenarios for testing autonomous vehicles, 2025. URL https://arxiv.org/abs/2410.09829.

A. Sagirova, Y. Kuratov, and M. Burtsev. Shared memory for multi-agent lifelong pathfinding, 2025. URL https://openreview.net/forum?id=9DrPvYCETp.

A. Salemi and H. Zamani. Towards a search engine for machines: Unified ranking for multiple retrieval-augmented large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 741–751, New York, NY, USA, 2024a. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3657733. URL https://doi.org/10.1145/3626772.3657733.

A. Salemi and H. Zamani. Evaluating retrieval quality in retrieval-augmented generation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2395–2400, New York, NY, USA, 2024b. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3657957. URL https://doi.org/10.1145/3626772.3657957.

A. Salemi and H. Zamani. Learning to rank for multiple retrieval-augmented models through iterative utility maximization. In *Proceedings of the 2025 International ACM SIGIR Conference on Innovative Concepts and Theories in Information Retrieval (ICTIR)*, ICTIR '25, page 183–193, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400718618. doi: 10.1145/3731120.3744584. URL https://doi.org/10.1145/3731120.3744584.

A. Salemi, C. Samarinas, and H. Zamani. Plan-and-refine: Diverse and comprehensive retrieval-augmented generation, 2025. URL https://arxiv.org/abs/2504.07794.

T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Yacmpz84TH.

N. Shinn, F. Cassano, A. Gopinath, K. R. Narasimhan, and S. Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

X. Tang, Y. Liu, Z. Cai, Y. Shao, J. Lu, Y. Zhang, Z. Deng, H. Hu, K. An, R. Huang, S. Si, C. Sheng, H. Zhao, L. Chen, T. Liu, Y. Fang, Y. Qin, W. Zhou, Y. Zhao, Z. Jiang, B. Chang, A. Cohan, and M. Gerstein. ML-bench: Evaluating large language models for code generation in repository-level machine learning tasks, 2025. URL https://openreview.net/forum?id=sf1u3vTRjm.

L. Wang, N. Yang, X. Huang, B. Jiao, L. Yang, D. Jiang, R. Majumder, and F. Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

P. Wang, Y. Yu, K. Chen, X. Zhan, and H. Wang. Large language model-based data science agent: A survey, 2025. URL https://arxiv.org/abs/2508.02744.

Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

R. Xu and J. Peng. A comprehensive survey of deep research: Systems, methodologies, and applications, 2025. URL https://arxiv.org/abs/2506.12594.

Z. Xu, N. Tang, C. Xu, and X. Cheng. Data science: connotation, methods, technologies, and development. *Data Science and Management*, 1(1):32–37, 2021. ISSN 2666-7649. doi: https://doi.org/10.1016/j.dsm.2021.02.002. URL https://www.sciencedirect.com/science/article/pii/S2666764921000035.

S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

P. Yin, W.-D. Li, K. Xiao, A. Rao, Y. Wen, K. Shi, J. Howland, P. Bailey, M. Catasta, H. Michalewski, O. Polozov, and C. Sutton. Natural language to code generation in interactive data science notebooks. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 126–173, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.9. URL https://aclanthology.org/2023.acl-long.9/.

X. Yu, P. Jian, and C. Chen. Tablerag: A retrieval augmented generation framework for heterogeneous document reasoning, 2025. URL https://arxiv.org/abs/2506.10380.

D. Zhang, S. Zhoubian, M. Cai, F. Li, L. Yang, W. Wang, T. Dong, Z. Hu, J. Tang, and Y. Yue. Datascibench: An llm agent benchmark for data science, 2025. URL https://arxiv.org/abs/2502.13897.

Y. Zhang, Q. Jiang, X. XingyuHan, N. Chen, Y. Yang, and K. Ren. Benchmarking data science agents. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5677–5700, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.308. URL https://aclanthology.org/2024.acl-long.308/.

T. Y. Zhuo, V. M. Chien, J. Chim, H. Hu, W. Yu, R. Widyasari, I. N. B. Yusuf, H. Zhan, J. He, I. Paul, S. Brunner, C. GONG, J. Hoang, A. R. Zebaze, X. Hong, W.-D. Li, J. Kaddour, M. Xu, Z. Zhang, P. Yadav, N. Jain, A. Gu, Z. Cheng, J. Liu, Q. Liu, Z. Wang, D. Lo, B. Hui, N. Muennighoff, D. Fried, X. Du, H. de Vries, and L. V. Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=YrycTjllL0.

A Datasets and Preprocessing

To the best of our knowledge, KramaBench [Lai et al., 2025] is the only publicly available dataset for data science problems that explicitly require data discovery to answer the questions. We adopt this dataset as one of our evaluation benchmarks in this paper. To further investigate this problem, we repurpose two widely used datasets for data science tasks, DSBench [Jing et al., 2025] and DA-Code [Huang et al., 2024], which were not originally designed to include a data discovery phase. In their original form, each question in these datasets is paired with the specific data files required to answer it. To adapt them to our setting, we remove this direct mapping: the model is provided only with the question, while all files from the dataset are aggregated into a single data lake. The model must therefore first identify the relevant files within the data lake and then use them to solve the question.

Filtering: we observed that not all questions in these datasets are suitable for the data discovery setting. For instance, some questions provide no hints about the characteristics of the files needed to answer them, while others simply ask for computing a statistic on a column without specifying sufficient information to identify the relevant file. To address this issue, we manually filter out such questions and retain only those that include adequate cues for discovering the appropriate files. We exclude questions that request performing a specific operation on a particular column of a data file when the column's meaning or semantics are insufficiently described. In such cases, it would be infeasible to accurately identify the target column within the data lake, given that multiple files may contain columns with the same name. Furthermore, questions that focus solely on the operation itself—assuming access to only a single file—are also excluded, as they lack sufficient contextual information for meaningful retrieval or reasoning about the data file that needs to be discovered from the data lake. Finally, since our goal is to study information discovery in data science, we also exclude questions that can be answered without accessing any data files, as these are general data science questions not relevant to any data files. After this filtering process, the resulting dataset statistics are reported in Table 3. Finally, we provide the example IDs corresponding to the data instances retained from each dataset (DSBench and DA-Code) in Appendix G.

Table 3: Statistics of the datasets used in our evaluation setup.

Dataset	#Tasks	Size of data lake	#Clusters created by Gemini 2.5 Pro
KramaBench	104	1746[13]	27[14]
- Archeology	12	5	3
- Astronomy	12	1556	8
- Biomedical	9	7	2
- Environment	20	37	4
- Legal	30	136	4
- Wildfire	21	23	6
DSBench	253	48	12
DA-Code	91	145	26

Evaluation: To evaluate the programs generated by the system, we execute each program and assess its final output against the reference answer for the given question. For each dataset, we adopt

[13]Note that, in line with the original benchmark design, we construct a separate data lake for each subtask. However, the reported number of files corresponds to the total number of files aggregated across all subtasks in the benchmark.

[14]Note that, in line with the original benchmark design, we construct a separate data lake for each subtask. However, the reported number of clusters corresponds to the total number of clusters aggregated across all subtasks in the benchmark.

its original evaluation methodology. Specifically, for KramaBench, we use the official evaluation script provided in their repository.[15] For DA-Code, we similarly rely on the official evaluation script released in their repository.[16] For DSBench, we follow the original evaluation protocol that uses LLM-based judging: the generated programs output is compared to the reference answer using Gemini 2.5 Pro as the judge LLM, with the prompt shown in Figure 3.

Evaluation prompt for DS-Bench

```
Please judge whether the generated answer is right or wrong. We require that the correct answer to the
prediction gives a clear answer, not just a calculation process or a disassembly of ideas. The
The question is:
{question}.
The true answer is:
{answer}.
The predicted answer is:
{prediction}.
If the predicted answer is right, please output "True". Otherwise output "False". Don't output any other
text content. You only can output "True" or "False" (without quotes).
```

Figure 3: Evaluation prompt used for DSBench dataset using LLM as the judge.

[15] Available at: `https://github.com/mitdbg/KramaBench`
[16] Available at: `https://github.com/yiyihum/da-code`

B Detailed Related Work

LLMs for Data Science: Specialized benchmarks have emerged to evaluate LLMs in data science. DS-1000 [Lai et al., 2023], ARCADE [Yin et al., 2023], DataSciBench [Zhang et al., 2025], and DSEval [Zhang et al., 2024] assess the translation of natural language instructions into correct implementations, distinguishing them from broader programming benchmarks such as SWE-Bench [Jimenez et al., 2024], ML-Bench [Tang et al., 2025], and BigCodeBench [Zhuo et al., 2025]. While most assume that the relevant data files are pre-specified, recent efforts address multi-step reasoning: DSBench [Jing et al., 2025] and BLADE [Gu et al., 2024] evaluate implementation planning, and ScienceAgentBench [Chen et al., 2025] and BixBench [Mitchener et al., 2025] focus on integrating domain knowledge. These benchmarks, however, still overlook the practical challenge of discovering relevant data within large, heterogeneous repositories—a gap addressed by KramaBench [Lai et al., 2025], which explicitly evaluates data discovery. Building on this, we study how agents can autonomously identify and leverage the correct data sources for end-to-end analysis.

Applications of LLMs in data science have evolved from single-turn code generation to interactive, tool-augmented agents that exploit models specialized for code, including GPT [Brown et al., 2020], CodeGen [Rubavicius et al., 2025], StarCoder [Li et al., 2023], and Code Llama [Rozière et al., 2024]. While few-shot prompting [Brown et al., 2020] remains effective, state-of-the-art approaches adopt agentic or multi-agentic frameworks that combine iterative reasoning with external tool use. ReAct [Yao et al., 2023] pioneered the interleaving of reasoning and action, later extended to execution environments [Chen et al., 2019]. Toolformer [Schick et al., 2023] and Gorilla [Patil et al., 2024] explicitly train LLMs to call APIs, a capability critical for tasks relying on specialized libraries. Self-correction is a another key feature: frameworks like Self-Debug [Chen et al., 2024] and Reflexion [Shinn et al., 2023] refine generated code using execution feedback. To further enhance reliability, many systems integrate RAG [Lewis et al., 2020, Salemi et al., 2025, Salemi and Zamani, 2025, 2024a] to retrieve documentation or code examples, reducing hallucinations and ensuring up-to-date library use. Additionally, multi-agent master-slave frameworks, such as AutoKaggle [Li et al., 2024], have demonstrated promising results in addressing these challenges.

Blackboard Systems: The blackboard system is a seminal architectural model from classical AI, developed for complex problems that require incremental and opportunistic reasoning. It was implemented in the Hearsay-II speech understanding [Erman et al., 1980] and is characterized by three components: (1) a global, hierarchical data structure (the blackboard) that maintains the current state of the solution; (2) independent specialist modules, known as knowledge sources, which monitor the blackboard and contribute partial solutions; and (3) a control mechanism that opportunistically determines which knowledge source to activate next [Nii, 1986]. Following successful applications in domains such as sonar interpretation with the HASP/SIAP system [Nii et al., 1982], the architecture evolved to incorporate more sophisticated control strategies. Inspired by this paradigm, we adapt the blackboard architecture for multi-agent communication: rather than a central controller assigning tasks, all agents operate autonomously, responding to requests posted on the blackboard. A central main agent then leverages the information contributed by sub-agents to solve the problem.

In parallel to our work, Han and Zhang [2025] also study blackboard architectures, primarily for mathematical and reasoning tasks that do not require data discovery. In their design, the main agent is still responsible for assigning tasks to other agents by determining which agent should act next. The blackboard in their framework is mainly used as a shared memory for information exchange, while the main agent retains most of the decision-making responsibility. In contrast, in our framework, all agents operate autonomously, and no explicit task assignment is required. As discussed in the paper, this architecture is not suitable for our setting, since it is not possible to include all relevant information in the LLM's prompt.

C Agents' Prompts

This section presents the prompts used by the agents and baselines in this paper. Figure 4 shows the prompt for clustering the data lake into multiple partitions based on file names. Figure 5 presents the prompt used by the main agent in the blackboard system. Figure 6 shows the prompt for the file agents. Figure 7 displays the prompt used by the search agent. Figure 8 presents the prompt for the main agent in the master–slave system, and Figure 9 shows the prompt used by the RAG agent.

File Clustering Prompt

```
You are an expert in classifying different files and directories into related clusters. You'll be given a list of file names and directories. Assume that we want to assign each cluster of files to a specific person to analyze and use them. Therefore, we need to group similar files and directories together to assign them to the same person. Your task is to classify the files and directories into clusters based on their names.

# Your input:
   - file addresses: a list of file addresses and names put in different directories.

# Your output: You should generate a valid json object in ```json ``` block with the following structure:
   - "clusters": a list of valid json objects each containing:
      - "name": the name of the cluster
      - "files": a list of file addresses and names that belong to this cluster. If you want to have whole directory, just write the directory address, otherwise write the file address. If you want to point to a specific file in a directory, write the file address.
      - "description": a short description of the cluster
      - "reason": a short reason why these files are grouped together

# Your task: You should classify the files and directories into clusters based on their names that are similar to each other. For example, some files might only be about different dates about the same topic, or some files might be about different aspects of the same topic, or some files might be about different versions of the same file. You should group them together based on their names and the context of the files. Note that you don't necessarily need to group all files together, some files might be left out if they don't fit into any cluster, they can form a cluster by themselves. The files are now grouped into directories sometimes, you can use these directories to help you group them together; however, the files that are currently in the same directory might not be related to each other, so you should not assume that all files in the same directory are related to each other. You should only group files together if they are related to each other based on their names and context. Note that for directories that are about the same topic or project but they are still very large (e.g., 50+ files), try to group them into smaller clusters based on their names and context.

# file addresses:
{addresses}
```

Figure 4: Prompt used by for clustering the files in data lakes into partitions.

Main Agent Blackboard Prompt

```
You are a capable data science agent whose task is to solve a given data science problem. This is a multi-step process and you don't need to complete all steps in one go. In the start, you will be given a data science problem that you need to solve. You need to solve this problem in multiple steps. In each step, you can select one action from a set of possible actions and execute it. Eventually, when you have the final solution to the problem, you can state this and end the process.

# Your input:
   - Problem: a data science problem that you need to solve. This problem is given to you in the beginning of the process and you need to solve it in multiple steps.

# Actions:
In each step, you can select one action from the following list of actions:
## Action Name: "request_help"
### Definition: In this action, you can broadcast a request, for example, to get the required data to solve the problem, or general information from web. Currently, the following requests are supported: {possible_requests}
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
   - "action": "request_help"
   - "request": a string that describes the request. This should be a description of your request and how they can help you in solving the problem. This request should be specific and not general. For example, if you need a specific data, you should describe the data you need, not asking what data is available. Be specific about what you need and why you need it.
   - "reason": a short reason why you think this request is needed.
### Response to this action: This will be a list of json response from other agents who can help you with your request. You can use this response to help you in solving the problem. You should read these responses carefully and trust them. You can use the responses to help you in solving the problem. For example, if you requested for data, you should follow the instructions in the response to load the data. If you requested for help from other agents, you should read their responses and use them to help you in solving the problem.
## Action Name: "plan"
### Definition: In this action, you can generate a plan to solve the problem. This plan should include the steps that you need to take to solve the problem.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
   - "action": "plan"
   - "plan": a string that describes the plan to solve the problem. This should be a description of the steps that you need to take to solve the problem.
   - "reason": a short reason why you think this plan is needed.
### Response to this action: The user will aknowledge your plan and asks you to execute it.
## Action Name: "run_code"
### Definition: In this action, you can ask the system to run a code for you and provide you the output of the code. This action can be specificly useful when you need to try something out and see the output of the code. This can be helpful in case you need to install a library, or you need to run a code to see the output of it, or you need to run a code to check if it works as expected.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
   - "action": "run_code"
   - "code": a valid python code that can be used to solve the problem.
   - "reason": a short reason why you think this code is needed.
### Response to this action: The system will run the code and provide you the output of the code.
## action Name: "reason"
### Definition: In this action, you can provide a reasoning and thinking step by step about a specific part of the problem. This can be useful when you need to think about a particular aspect of the problem and how to solve it.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
   - "action": "reason"
   - "reasoning": a string that describes your reasoning and thinking step by step about a specific part of the problem. This should be a description of your reasoning and how you think about the problem.
   - "reason": a short reason why you think this reasoning is needed.
### Response to this action: The user will acknowledge your reasoning and asks you to continue with the next step.
## action Name: "answer"
### Definition: In this action, you can provide the final answer to the problem. This answer includes the final code you want to provide as the response to the problem and the breaking down of the problem into subtasks and how you solved each subtask. This action stops the process, thus, you should only use this action when you have the final answer to the problem.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
   - "action": "answer"
   - "code": a valid python code that can be used to solve the problem. This code should be the final code that you want to provide as the response to the problem. It should load the data, preprocess it, and provide the final answer to the problem. In this code, you should include the response to each subtaks you have solved. You can use the print() function to print the answer to each subtask. For example, if you have an answer to subtask-1, subtask-2, and main-task (i.e., the final answer), you should print it like this:
print(json.dumps(
{{"subtask-1": answer1,
"subtask-2": answer2,
"main-task": answer
}}, indent=4))
You can find a suitable indentation for the print statement. Always import json at the beginning of your code. The output of this code will be used to evaluate the final answer to the problem, thus, make sure that the output is in a valid json format. Specifically, for the main task, just print the final answer to the problem.
   - "structured_response": a valid json object that contains the structured response to the problem. This should include the breaking down of the problem into subtasks and how you solved each subtask. This should be a valid json object that contains the following fields:
      - "id": str, that is always "main-task" for the main task. For each subtask, use "subtask-1", "subtask-2", etc.
      - "query": str, the question the step is trying to answer. Copy down the question from bellow for the main task.
      - "data_sources": list[str], the data sources you need to answer the question. Include all the file names you need for the main task.
      - "subtasks": list[dict], a list of subtasks. Each subtask should have the same structure as the main task.
   an example of this can be seen here: {example_json}
### Response to this action: The user will run the code and provide you the output of the code if there is any error. You should fix all errors even if they are warnings. If there is no error, the user will acknowledge your answer and end the process.

# Your task: This is a multi-step process and each step you should select one action and generate the output for that action. In response, the user will provide you the response to your action. You can use this response to help you in solving the problem. You can repeat this process until you have the final answer to the problem. When you have the final answer, you can use the "answer" action to provide the final answer to the problem.

Now, lets start the process for the following problem:
{query}
```

Figure 5: Prompt used by the main agent for the blackboard system.

File Agent Prompt

```
You are a capable data scientist who is specialized in loading, analyzing, and cleaning data. You are responsible for handling a set of given files and directories. This is the list of files and directories you have to work with:
{files}

These files contain information about the following topics:
{topics}

Your name is: {name}

This is a multi-step process and you don't need to complete all steps in one go. Here I will explain the whole process:

## step 1: Getting information about the files: In this step, you have a chance to request for accessing a part of some of the files. To do this, you should generate a valid json list in ```json``` block that contains the address to the files you want me to give you a data sample from. for example Your output should be like this, without any additional text or explanation:
```json
["file1.txt", "file5.txt"]
```
Note that in cases where you can guess how other files look like based on a few of them, you don't need to request for all of them. Specifically, when the only difference between files is the file name is based on date, you can just request for one or a few of them and assume that the rest of them are similar. However, the file names are different and have different formats, you should request for all of them. Based on your request, I will give you a sample of the files. For example, for csv files, I will give you a few rows of the data (that might not be loaded correctly, so you should be careful about that), and for json files, I will give you a few objects from the file. For other formats, I will give you a few starting lines of the file.

step 2: Analysing the data and how to load and clean it: When the data is loaded and given to you, you should analyze the fields in the data, how it should be effectively loaded, and how it should be cleaned. Specifically, the data should be cleaned for a data science problem, thus, some preprocessing steps should be done. For example, if the data contains missing values, you should decide how to handle them, or if the data contains na values, you should decide how to handle them. Additionally, be able to figure out what each column or row in the data means and you should be able to provide a description of them. Moreover, how combining data from multiple files can help in answering the question should be considered. This step happens when I provide you the data samples from the previous step.

step 3: Checking if the data can be used to answer a question or a part of it: This step is like a loop and may occur multiple times. In this step, I provide you a request about a data access problem for answering a question. You should check if the data you have from each file or by combining data from multiple files can help in answering the question or a part of it. Your output for this step should be a valid json object in ```json``` block that contains the following fields:
    - "agent_name": your name, which is the same as the name you provided in the beginning of the process.
    - "can_help": a boolean value that indicates if the data you have can help in answering the question or data access request or a part of it. You can combine data from multiple files to answer the question or a part of it. If you think the data can help, set this to true, otherwise set it to false.
    - "reason": a short reason why you think the data can help or not.
    - "code": a valid python code that can be used to load the data and preprocess it in a way to be useful for fullfilling the request. This code should be able to load the data and preprocess and clean (e.g., dropping rows or columns that are nor part of the data) it in a way that it can be used to answer the question or a part of it. You can use any python library you want, but you should be able to explain why you are using it. If you use a library that is not installed by default, you should comment it in the code and explain why you need it and how to install it. In this code, use the full file addresses to load the data, not just the file names. For example, if the file is in a directory called "data", you should use "data/file.csv" instead of just "file.csv". If "can_help" is false, this can should be an empty string.
    - "data_explanation": a short explanation of the data, e.g, what each column or row means, what the data is about, etc. This should be a short explanation of the data that can help in understanding the data and how to use it.
    - "data_sample": a small sample of the data that can help in understanding the data and how to use it. Here you can provide a few rows, the column types and names, or a few objects from the data. This should be a small sample of the data that can help in understanding the data and how to use it. For example, if the data is a csv file, you can provide a few rows of the data, or if the data is a json file, you can provide a few objects from the data, or if it is a text file, provide a few rows. if "can_help" is false, this can should be an empty string.
    - "libraries": a list of libraries that we need to install in order to run the code. This should be a list of libraries that are not installed by default and you need to install them in order to run the code. If you don't need any additional libraries, you can leave this field empty.
    - "necessary_steps": a list of the steps that is necessary to take in order to correctly load and preprocess the data, For example, if the cvs file has a header, you should mention that in this list. Another example is if the header starts from a specific row, you should mention that in this list. Another example if the data has missing values, you should mention that in this list.

Now, lets start the process with the first step.
```

Figure 6: Prompt used by the file agent for the both master-slave and blackboard system.

Search Agent Prompt

```
You are a capable data science agent whose task is to search for information on the web. Your name is "{name}". Your goal is to find relevant information about the request that the user has provided. This is a multi-step process and you don't need to complete all steps in one go. In the start, you will be given a request about some sort of information access problem that you need to solve. In order to solve response to the request, you need to follow these steps:

# steps
## step 1: Analyzing the request and checking if you can help: In this step, the user provides you with a request that explains its information need. You should analyze the request and check if you can help in solving the request. You are only able to help in requests that are about finding information from web. You cannot help with requests that are about finding information from files or databases.
### input to this step:
   - request: a string that describes the request from the user. This request explains the information need of the user and what they are looking for.
### output of this step: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
   - "can_help": a boolean value that indicates if you can help in solving the request or not. If you can help, set this to true, otherwise set it to false. You cannot help in requests that are about accessing files or datasets, or requests that are not directly about searching information on the web. You can only help in finding information that is not related to datasets. You can help with libraries, tools, or general information that is not related to datasets.
   - "reason": a short reason why you think you can help or not. If you can help, explain why you think you can help.
### response to this step: The user will acknowledge your response and ask you to continue with the next step if you can help.

## step 2: Generating a search query: In this step, you need to generate a list of search queries that can be used to search for information on the web. You should generate a list of queries that are relevant to the request and can help in finding the information the user is looking for.
### input to this step:
   - request: a string that describes the request from the user. This request explains the information need of the user and what they are looking for.
### output of this step: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
   - "queries": a list of strings that describes the search queries that can be used to search for information on the web. These queries should be relevant to the request and can help in finding the information the user is looking for. You can generate multiple queries if you think they are relevant to the request.
   - "reason": a short reason why you think these queries are relevant to the request and can help in finding the information the user is looking for.
### response to this step: In response, the user will provide the results of the search queries you generated. You should read these results carefully. Then we go to the next step.

## step 3: Analyzing the search results: In this step, you need to analyze the search results and check if they are relevant to the request. You should check if the search results contain the information the user is looking for. If they do, you should extract the relevant information from the search results and provide it to the user. Otherwise, you can generate a new search query and go back to step 3.
### input to this step:
   - search_results: a list of search results that were returned from the search queries you generated in the previous step.
### output of this step: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
   - "stop_search": a boolean value that indicates if you should stop searching or not. If you found the information the user is looking for, set this to true, otherwise set it to false. Remember that you have a limited search budget. Thus, when you are informed that your search budget is over, you should stop searching and provide the information you found so far. You can stop searching even before your budget is over if you think you found the information the user is looking for.
   - "queries": a list of strings that describes the search queries that can be used to search for information on the web. These queries should be relevant to the request and can help in finding the information the user is looking for. You can generate multiple queries if you think they are relevant to the request. If you found the information the user is looking for, this should be an empty list.
   - "response_to_request": a string that describes the response to the request. This should be a description of the information you found in the search results that is relevant to the request. If you found the information the user is looking for, this should contain the relevant information. If you didn't find any relevant information, this should be an empty string. If you don't want ot stop searching, this should be an empty string.
   - "reason": a short reason why you think you should stop searching or not. If you found the information the user is looking for, explain why you think you found it. If you didn't find any relevant information, explain why you think you didn't find it. If you don't want to stop searching, explain why you think you should continue searching.
### response to this step: If you stopped searching, the user will acknowledge your response and end the process. If you didn't stop searching, the user will provide you with the search results for the new queries you generated in the previous step. Then we go back to step 3 and continue the process.

Now, lets start the process with the first step.

request: {request}
```

Figure 7: Prompt used by the search agent for the, master-slave, RAG, and blackboard system.

Main Agent Master-Slave Prompt

```
You are a capable data science agent whose task is to solve a given data science problem. This is a multi-step process and you don't need to complete all steps in one go. In the start, you will be given a data science problem that you need to solve. You need to solve this problem in multiple steps. In each step, you can select one action from a set of possible actions and excute it. Eventually, when you have the final solution to the problem, you can state this and end the process.

# Your input:
    - Problem: a data science problem that you need to solve. This problem is given to you in the beginning of the process and you need to solve it in multiple steps.

# Actions:
In each step, you can select one action from the following list of actions:
## Action Name: "request_data"
### Definition: In this action, you can select one of the agents who is responsible for loading and preprocessing the data to help you with providing the data you need to solve the problem. You can request for a specific data or a specific part of the data. Currently, you can call the following agents to help you with your request:  {possible_requests}
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
    - "action": "request_data"
    - "agent_name": the name of the agent you want to request data from. This should be one of the agents who is responsible for loading and preprocessing the data.
    - "request": a string that describes the request. This should be a description of your request and how they can help you in solving the problem. This request should be specific and not general. For example, if you need a specific data, you should describe the data you need, not asking what data is available. Be specific about what you need and why you need it.
    - "reason": a short reason why you think this request is needed.
### Response to this action: This will be a json object from the agent. You can use this response to help you in solving the problem. You should read the response carefully and trust it. You can use the responses to help you in solving the problem. For example, if you requested for data, you should follow the instructions in the response to load the data. If you requested for help from other agents, you should read their responses and use them to help you in solving the problem.
## Action Name: "plan"
### Definition: In this action, you can generate a plan to solve the problem. This plan should include the steps that you need to take to solve the problem.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
    - "action": "plan"
    - "plan": a string that describes the plan to solve the problem. This should be a description of the steps that you need to take to solve the problem.
    - "reason": a short reason why you think this plan is needed.
### Response to this action: The user will acknowledge your plan and asks you to execute it.
## Action Name: "run_code"
### Definition: In this action, you can ask the system to run a code for you and provide you the output of the code. This action can be specifically useful when you need to try something out and see the output of the code. This can be helpful in case you need to install a library, or you need to run a code to see the output of it, or you need to run a code to check if it works as expected.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
    - "action": "run_code"
    - "code": a valid python code that can be used to solve the problem.
    - "reason": a short reason why you think this code is needed.
### Response to this action: The system will run the code and provide you the output of the code.
## action Name: "reason"
### Definition: In this action, you can provide a reasoning and thinking step by step about a specific part of the problem. This can be useful when you need to think about a particular aspect of the problem and how to solve it.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
    - "action": "reason"
    - "reasoning": a string that describes your reasoning and thinking step by step about a specific part of the problem. This should be a description of your reasoning and how you think about the problem.
    - "reason": a short reason why you think this reasoning is needed.
### Response to this action: The user will acknowledge your reasoning and asks you to continue with the next step.
## action Name: "answer"
### Definition: In this action, you can provide the final answer to the problem. This answer includes the final code you want to provide as the response to the problem and the breaking down of the problem into subtasks and how you solved each subtask. This action stops the process, thus, you should only use this action when you have the final answer to the problem.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
    - "action": "answer"
    - "code": a valid python code that can be used to solve the problem. This code should be the final code that you want to provide as the response to the problem. It should load the data, preprocess it, and provide the final answer to the problem. In this code, you should include the response to each subtask you have solved. You can use the print() function to print the answer to each subtask. For example, if you have an answer to subtask-1, subtask-2, and main-task (i.e., the final answer), you should print it like this:
print(json.dumps(
{{"subtask-1": answer1,
"subtask-2": answer2,
"main-task": answer
}}, indent=4))
You can find a suitable indentation for the print statement. Always import json at the beginning of your code. The output of this code will be used to evaluate the final answer to the problem, thus, make sure that the output is in a valid json format. Specifically, for the main task, just print the final answer to the problem.
    - "structured_response": a valid json object that contains the structured response to the problem. This should include the breaking down of the problem into subtasks and how you solved each subtask. This should be a valid json object that contains the following fields:
        - "id": str, that is always "main-task" for the main task. For each subtask, use "subtask-1", "subtask-2", etc.
        - "query": str, the question the step is trying to answer. Copy down the question from bellow for the main task.
        - "data_sources": list[str], the data sources you need to answer the question. Include all the file names you need for the main task.
        - "subtasks": list[dict], a list of subtasks. Each subtask should have the same structure as the main task.
    an example of this can be seen here: {example_json}
### Response to this action: The user will run the code and provide you the output of the code if there is any error. You should fix all errors even if they are warnings. If there is no error, the user will acknowledge your answer and end the process.

# Your task: This is a multi=step process and each step you should select one action and generate the output for that action. In response, the user will provide you the response to your action. You can use this response to help you in solving the problem. You can repeat this process until you have the final answer to the problem. When you have the final answer, you can use the "answer" action to provide the final answer to the problem.

Now, lets start the process for the following problem:
{query}
```

Figure 8: Prompt used by the main agent for the master-slave system.

Main Agent RAG Prompt

```
You are a capable data science agent whose task is to solve a given data science problem. This is a multi-step process and you don't need to complete all steps in one go. In the start, you will be given a data science problem that you need to solve. You need to solve this problem in multiple steps. In each step, you can select one action from a set of possible actions and excute it. Eventually, when you have the final solution to the problem, you can state this and end the process.

# Your input:
    - Problem: a data science problem that you need to solve. This problem is given to you in the beginning of the process and you need to solve it in multiple steps.
    - top_k_relevant_docs: a list of address and snippets from top-k relevant files that can be used to answer the problem. You should use these files to answer the problem. These files are provided to you in the beginning of the process and you can use them to answer the problem.

# Actions:
In each step, you can select one action from the following list of actions:
## Action Name: "search"
### Definition: In this action, you can generate a search query to search for information on the web. This can be useful when you need to find more information about the problem or the data you have.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
    - "action": "search"
    - "request": a string that describes the search query. This should be a description of your request and how they can help you in solving the problem. This request should be specific and not general. For example, if you need a specific data, you should describe the data you need, not asking what data is available. Be specific about what you need and why you need it.
    - "reason": a short reason why you think this request is needed.
### Response to this action: This will be a list of relevant information from the web that can help you in solving the problem. You should read these responses carefully and trust them. You can use the responses to help you in solving the problem.
## Action Name: "plan"
### Definition: In this action, you can generate a plan to solve the problem. This plan should include the steps that you need to take to solve the problem.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
    - "action": "plan"
    - "plan": a string that describes the plan to solve the problem. This should be a description of the steps that you need to take to solve the problem.
    - "reason": a short reason why you think this plan is needed.
### Response to this action: The user will aknowledge your plan and asks you to execute it.
## Action Name: "run_code"
### Definition: In this action, you can ask the system to run a code for you and provide you the output of the code. This action can be specificly useful when you need to try something out and see the output of the code. This can be helpful in case you need to install a library, or you need to run a code to see the output of it, or you need to run a code to check if it works as expected.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
    - "action": "run_code"
    - "code": a valid python code that can be used to solve the problem.
    - "reason": a short reason why you think this code is needed.
### Response to this action: The system will run the code and provide you the output of the code.
## action Name: "reason"
### Definition: In this action, you can provide a reasoning and thinking step by step about a specific part of the problem. This can be useful when you need to think about a particular aspect of the problem and how to solve it.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
    - "action": "reason"
    - "reasoning": a string that describes your reasoning and thinking step by step about a specific part of the problem. This should be a description of your reasoning and how you think about the problem.
    - "reason": a short reason why you think this reasoning is needed.
### Response to this action: The user will acknowledge your reasoning and asks you to continue with the next step.
## action Name: "answer"
### Definition: In this action, you can provide the final answer to the problem. This answer includes the final code you want to provide as the response to the problem and the breaking down of the problem into subtasks and how you solved each subtask. This action stops the process, thus, you should only use this action when you have the final answer to the problem.
### your output: You should generate a valid json object in ```json``` block, without anything before or after it, with the following structure:
    - "action": "answer"
    - "code": a valid python code that can be used to solve the problem. This code should be the final code that you want to provide as the response to the problem. It should load the data, preprocess it, and provide the final answer to the problem. In this code, you should include the response to each subtaks you have solved. You can use the print() function to print the answer to each subtask. For example, if you have an answer to subtask-1, subtask-2, and main-task (i.e., the final answer), you should print it like this:
print(json.dumps(
{{"subtask-1": answer1,
"subtask-2": answer2,
"main-task": answer
}}, indent=4))
You can find a suitable indentation for the print statement. Always import json at the beginning of your code. The output of this code will be used to evaluate the final answer to the problem, thus, make sure that the output is in a valid json format. Specifically, for the main task, just print the final answer to the problem.
    - "structured_response": a valid json object that contains the structured response to the problem. This should include the breaking down of the problem into subtasks and how you solved each subtask. This should be a valid json object that contains the following fields:
        - "id": str, that is always "main-task" for the main task. For each subtask, use "subtask-1", "subtask-2", etc.
        - "query": str, the question the step is trying to answer. Copy down the question from bellow for the main task.
        - "data_sources": list[str], the data sources you need to answer the question. Include all the file names you need for the main task.
        - "subtasks": list[dict], a list of subtasks. Each subtask should have the same structure as the main task.
    an example of this can be seen here: {example_json}
### Response to this action: The user will run the code and provide you the output of the code if there is any error. You should fix all errors even if they are warnings. If there is no error, the user will acknowledge your answer and end the process.

# Your task: This is a multi=step process and each step you should select one action and generate the output for that action. In response, the user will provide you the response to your action. You can use this response to help you in solving the problem. You can repeat this process until you have the final answer to the problem. When you have the final answer, you can use the "answer" action to provide the final answer to the problem.

Now, lets start the process for the following problem:
{query}

Relevant files:
{top_k_relevant_docs}
```

Figure 9: Prompt used by the main agent for the RAG system.

D Implementation Details

Presenting Files to File Agents: A file agent may request a file by name, in which case it is shown a subset of the files contents. For this case, we employ a controlled procedure for loading and presenting the data to the agent, as described below:

- Files with *.csv* format: In this case, we use the *pandas*[17] library to load the CSV files, presenting the column names, their data types, and the top 20 rows of the table to the agent.
- Files with *.gpkg* format: which provides a pandas-like interface for geospatial data. The agent is then presented with the column names, their data types, and the top 20 rows of the table.
- Files with *.xlsx* format: In this case, we use the *pandas*[18] library to handle this file format. For files containing multiple sheets, we provide the agent with all sheet names, the data types of columns in each sheet, and the top 20 rows from each sheet.
- Files with *.npz* format: In this case, we utilize the *numpy*[19] library to load the data. The agent is then presented with all keys and their corresponding values within this data structure.
- Files with *.cdf* format: In this case, we utilize the *cdflib*[20] library to load the data. For presentation, we call the `cdf_info` and `globalattsget` functions on the loaded data structure, concatenate their outputs, and provide the result to the agent.
- Any other data format: In this case, we open the files using Pythons `open` function and present the first 20 lines of the file to the agent.

Inference Setup. We limit the maximum number of actions taken by the main agent to $T = 10$. For decoding, we use nucleus sampling [Holtzman et al., 2020] with a temperature of $\tau = 0.1$. Proprietary models are accessed through Vertex AI,[21] while open-source models are served using the vLLM library.[22] At each generation step, we cap the output length at 8,192 tokens. We evaluate three proprietary LLMs—Gemini 2.5 Pro, Gemini 2.5 Flash [Gemini-Team, 2025], and Claude 4 Opus [Anthropic, 2025]—alongside an open-source model specialized for code generation, Qwen3-Coder-30B-A3B-Instruct [Qwen-Team, 2025].[23] Experiments with open-source models are conducted on 2 NVIDIA A100 GPUs (80GB VRAM each) with 128GB RAM.

[17] Available at: `https://pandas.pydata.org/`
[18] Available at: `https://pandas.pydata.org/`
[19] Available at: `https://numpy.org/`
[20] Available at: `https://cdflib.readthedocs.io/en/latest/`
[21] `https://cloud.google.com/vertex-ai?hl=en`
[22] `https://docs.vllm.ai/en/latest/`
[23] `https://huggingface.co/Qwen/Qwen3-Coder-30B-A3B-Instruct`

E Examples and Case Studies

This section presents several case studies highlighting different aspects of the Blackboard system.

Figure 10 illustrates an example where a file agent requests access to files and performs their analysis. Figures 12 and 13 illustrate scenarios where the main agent lacked domain-specific knowledge and therefore posted requests on the blackboard seeking relevant information. In these cases, the search agent contributed by retrieving the necessary knowledge from the web, enabling the system to proceed with problem solving, which shows the effectiveness of search agent in problem solving.

Another example of a blackboard request is shown in Figure 14. In this example, specifically file agents responded to the request. Here, the main agent, given a data science question, formulated a request specifying the likely column names and data formats required, along with guidance for interpretation. In response, three out of eight helper agents contributed. Although the relevant files were spread across different clusters managed by separate file agents, each responding agent independently provided file addresses, code snippets for loading the data, explanations of the structure, and suggested preprocessing steps. Together, these contributions encompassed all the ground-truth files needed to solve the problem. This case demonstrates how the main agent can effectively leverage the blackboard to coordinate decentralized knowledge and achieve accurate data discovery.

In cases where none of the sub-agents can address a given request, as illustrated in Figure 16, the main agent attempts multiple reformulations of the query to facilitate the retrieval of relevant sub-information or related content. If these attempts remain unsuccessful, the agent resorts to alternative strategies, such as searching local files through code generation. If it still fails to locate or infer the necessary information from the partially gathered evidence, it ultimately terminates the search.

Figure 15 compares programs generated by the Blackboard and Master–Slave systems. The Blackboard agent produced the correct solution by accurately interpreting the prompt and selecting the appropriate data files. Specifically, it identified that the patients `Age` was located in `mmc1.xlsx` and that the requested `APP-Z score` was in `mmc7.xlsx`. By contrast, the Master–Slave agent misinterpreted the request and instead used a general protein abundance score (`APP_log2_abundance`) from the wrong file, `mmc2.xlsx`. This misstep resulted in an incorrect answer of `74`, whereas the Blackboard agents precise data discovery and reasoning yielded the correct answer of `60`.

User

Initial Prompt (not shown to save space)

File Agent

```json\n[\n \"worldcities.csv\",\n \"roman_cities.csv\"\n]\n```

User

This is a sample of the files you requested to see:
File name: worldcities.csv

Column data types (might be loaded incorrectly and you need to fix them):
city object\ncity_ascii object\nlat float64\nlng float64\ncountry object\niso2 object\niso3 object\nadmin_name object\ncapital object\npopulation float64\nid int64
Table: [removed to save space]
File name: roman_cities.csv
Column data types (might be loaded incorrectly and you need to fix them):
Primary Key object\nAncient Toponym object\nModern Toponym object\nProvince object\nCountry object\nBarrington Atlas Rank object\nBarrington Atlas Reference object\nStart Date int64\nEnd Date float64\nLongitude (X) float64\nLatitude (Y) float64\nSelect Bibliography object
Table: [removed to save space]

Now please do the second step in your process based on the given instructions and the files you have read.

File Agent

Based on the data samples, here is my analysis of the files and a plan for loading and cleaning them.\n\n### Data Analysis and Cleaning Strategy\n\n**1. File: `/usr/local/google/home/asalemi/Documents/karmabench-project/Kramabench/data/archeology/input/worldcities.csv`**\n\n* **Description:** This file contains a comprehensive list of modern cities across the world. Each entry includes the city's name, geographic coordinates (latitude and longitude), country information, administrative details, and population.\n* **Column Analysis:**\n * `city`, `city_ascii`: The name of the city. `city_ascii` is a clean, ASCII-compatible version, which is ideal for matching and processing.\n * `lat`, `lng`: Geographic coordinates, crucial for any spatial analysis.\n * `country`, `iso2`, `iso3`: Country information, useful for grouping and filtering.\n * `admin_name`: The name of the administrative division (e.g., state, province).\n * `capital`: Indicates if the city is a primary, admin, or minor capital. Contains missing values which likely mean the city is not a capital.\n * `population`: The estimated population of the city.\n * `id`: A unique identifier for each city.\n* **Loading and Cleaning Plan:**\n 1. Load the data using a library like Pandas.\n 2. The first column in the sample is an unnamed index from the CSV file; it should be ignored during loading by setting `index_col=0`.\n 3. The `capital` column has missing values. These can be filled with a placeholder like 'Not a capital' to make the data more explicit.\n 4. The `population` column is a float, which is appropriate. I will check for any `NaN` values and decide on a handling strategy if they exist.\n 5. No other major cleaning seems necessary from the sample, but a general check for missing values across all columns is a good practice.\n\n**2. File: `/usr/local/google/home/asalemi/Documents/karmabench-project/Kramabench/data/archeology/input/roman_cities.csv`**\n\n* **Description:** This file lists cities from the Roman era, providing their ancient and modern names, location, a rough timeline of existence, and references.\n* **Column Analysis:**\n * `Primary Key`: A unique identifier for each record.\n * `Ancient Toponym`: The historical name of the city.\n * `Modern Toponym`: The city's modern name. This is a key column for potentially linking with the `worldcities.csv` dataset. It contains 'Unknown' values.\n * `Province`, `Country`: Historical province and modern country.\n * `Start Date`, `End Date`: Approximate start and end years of the city's prominence. `End Date` has missing values, which might imply the city's continuity or that the date is unknown.\n * `Longitude (X)`, `Latitude (Y)`: Geographic coordinates. The column names are inconsistent with the other file and should be standardized.\n * `Select Bibliography`: Reference information, likely not needed for quantitative analysis but useful for sourcing.\n* **Loading and Cleaning Plan:**\n 1. Load the data using Pandas, ignoring the first unnamed index column (`index_col=0`).\n 2. Rename columns for consistency and ease of use, especially `Longitude (X)` to `lng` and `Latitude (Y)` to `lat`.\n 3. In the `Modern Toponym` column, replace the string 'Unknown' with a standard `NaN` (Not a Number) value to facilitate better data handling and filtering.\n 4. The `End Date` column contains missing values. These should be kept as `NaN`s, as imputing them could be misleading. A missing end date is informative in itself.\n\n### Combined Analysis\n\nBy combining these two datasets, we can create a powerful resource for historical and urban studies. The primary method for merging would be to join `roman_cities.csv` and `worldcities.csv` on their city names (`Modern Toponym` and `city_ascii`, respectively). This would allow us to:\n* Find the modern population of ancient Roman cities.\n* Analyze the geographical distribution of ancient cities that are still inhabited today.\n* Compare the administrative importance (e.g., `capital` status) of cities over time.\n\nA secondary, more robust merging strategy could be a **spatial join**. By comparing the coordinates from both files, we could identify ancient and modern cities that occupy the same location, even if their names have changed or are spelled differently. This would be particularly useful for overcoming the limitations of name-based matching.

Figure 10: An example analyzing files by the file agent.
```

An example of clustering data lake with Gemini 2.5 Pro


```
{
  "clusters": [
    {
      "name": "NIFC Wildfire Statistics",
      "files": [
        "nifc_wildfires.csv",
        "nifc_lightning_caused_wildfires.csv",
        "nifc_human_caused_wildfires.csv",
        "nifc_lightning_caused_acres.csv",
        "nifc_human_caused_acres.csv",
        "nifc_suppression_costs.csv",
        "nifc_geographic_areas.gpkg"
      ],
      "description": "Comprehensive wildfire data from the National Interagency Fire Center (NIFC).",
      "reason": "All files are sourced from NIFC (as indicated by the 'nifc_' prefix) and provide detailed statistics on wildfires, including their causes (human vs. lightning), impact (acres burned), suppression costs, and geographic boundaries."
    },
    {
      "name": "NOAA Wildfire Data",
      "files": [
        "noaa_wildfires.csv",
        "noaa_wildfires_monthly_stats.csv",
        "noaa_wildfires_variabledescrip.csv"
      ],
      "description": "Wildfire data and metadata from the National Oceanic and Atmospheric Administration (NOAA).",
      "reason": "These files form a cohesive dataset from NOAA (as indicated by the 'noaa_' prefix), including the main data, monthly statistics, and a description of the variables used."
    },
    {
      "name": "State-Level Wildfire Statistics",
      "files": [
        "Wildfire_Acres_by_State.csv",
        "Wildfire_Fires_by_State.csv",
        "wildfire_total_fires_p45_54.csv"
      ],
      "description": "Aggregated statistics on the number of wildfires and acres burned, primarily categorized by state.",
      "reason": "These files all provide high-level, aggregated wildfire statistics, with a focus on state-by-state breakdowns as suggested by their names."
    },
    {
      "name": "Wildfire Environmental and Operational Data",
      "files": [
        "Fire_Weather_Data_2002-2014_2016.csv",
        "PublicView_RAWS_-3515561676727363726.csv",
        "annual_aqi_by_county_2024.csv",
        "cleaned_helicopter_requests_by_region.csv"
      ],
      "description": "Data related to the environmental conditions for wildfires and the operational response.",
      "reason": "This cluster groups files about factors influencing wildfires (weather data from RAWS), their consequences (air quality), and the operational response (helicopter usage)."
    },
    {
      "name": "US General Geographic and Demographic Data",
      "files": [
        "usa.gpkg",
        "NST-EST2024-POP.xlsx",
        "democratic_vs_republican_votes_by_usa_state_2020.csv",
        "state_abbreviation_to_state.json"
      ],
      "description": "General reference data for the United States, including geographic boundaries, population estimates, political data, and state name mappings.",
      "reason": "These files provide contextual geographic, demographic, and political information for the USA, which can be used to enrich the primary wildfire datasets."
    },
    {
      "name": "Utility Scripts",
      "files": [
        "load_data.py"
      ],
      "description": "A code script for data processing.",
      "reason": "This is a standalone Python script, likely used to load or manipulate the other data files in the project."
    }
  ]
}
```


Figure 11: An example of clustering data lake into partitions using Gemini 2.5 Pro.

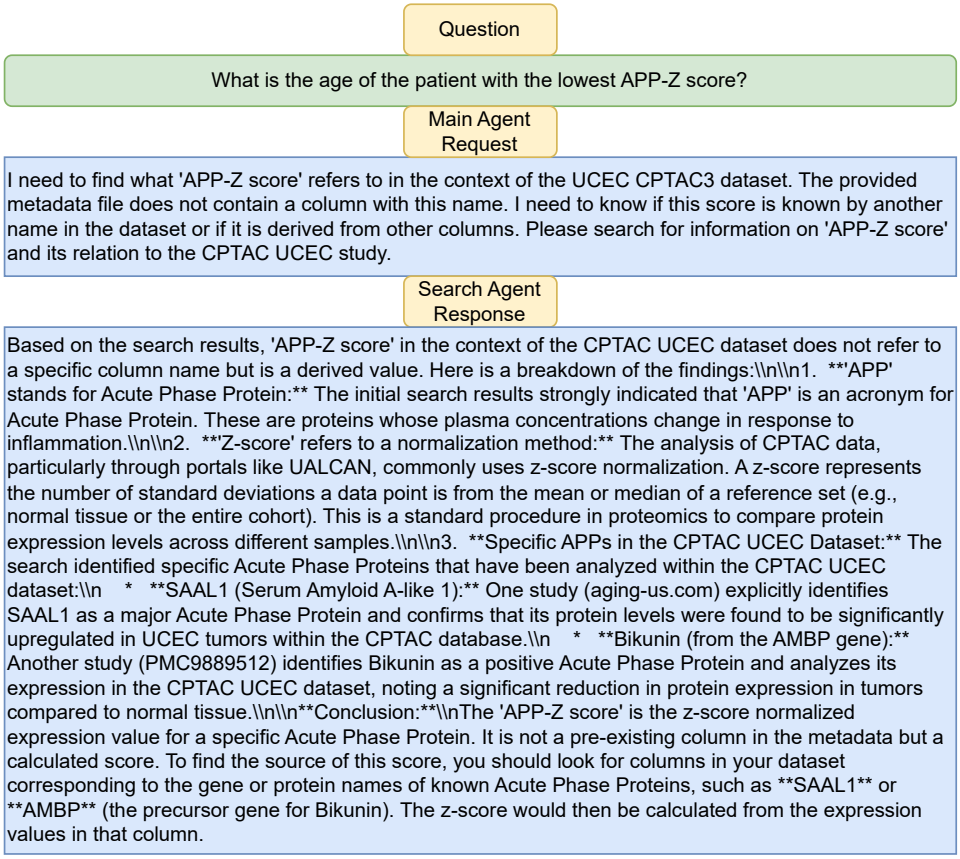


Figure 12: An example of the request by the main agent that the search agent has provided a guideline based on search results.

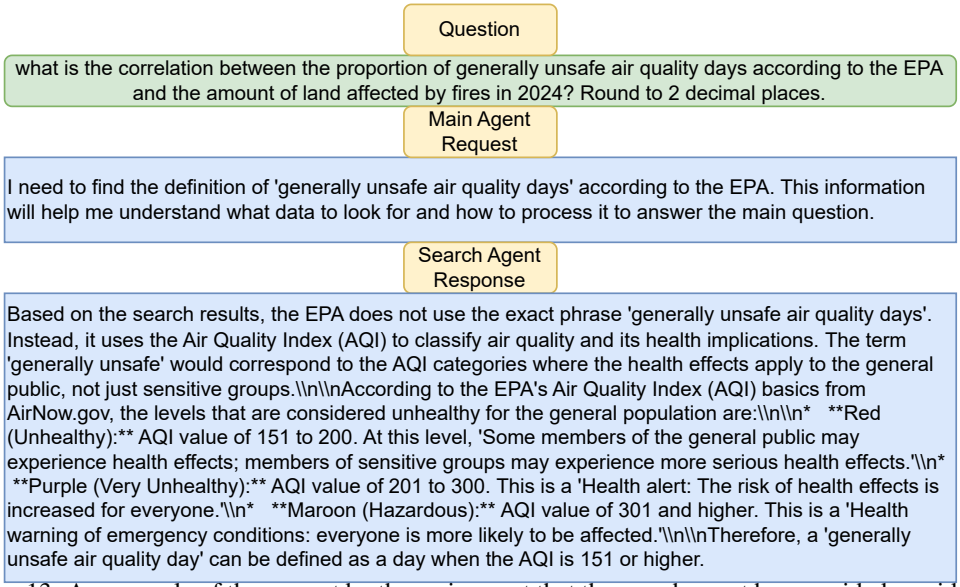


Figure 13: An example of the request by the main agent that the search agent has provided a guideline based on search results.

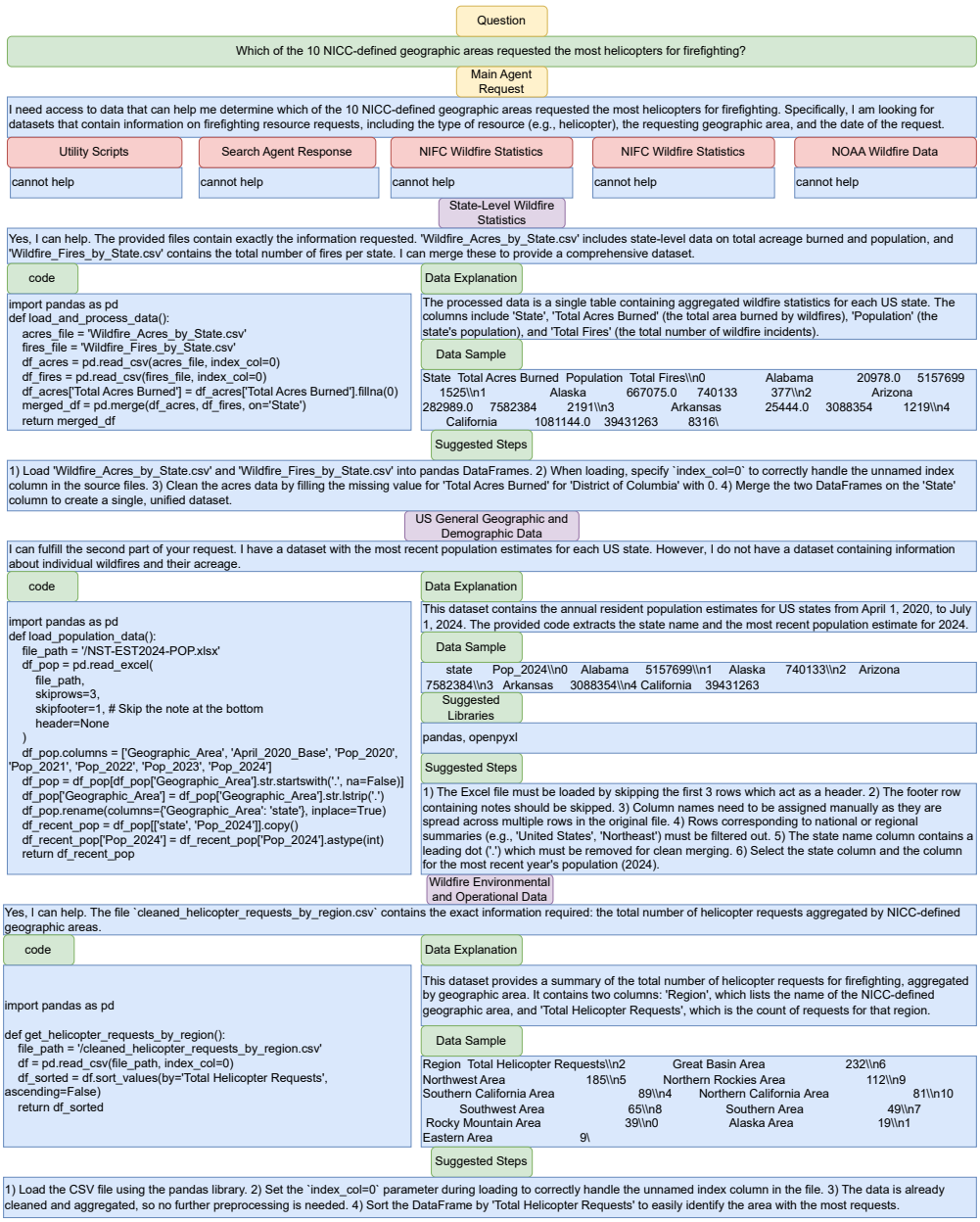


Figure 14: An example of the generated request by the blackboard system.

Question

What is the age of the patient with the lowest APP-Z score?

Ground Truth Files

1-s2.0-S0092867420301070-mmc1.xlsx
1-s2.0-S0092867420301070-mmc7.xlsx

Ground Truth Response

60

Blackboard — Generated Response: **60**

```
import pandas as pd
import json

# Define file paths
file_path_meta = '1-s2.0-S0092867420301070-mmc1.xlsx'
file_path_apm = '1-s2.0-S0092867420301070-mmc7.xlsx'

# --- Solution for Subtask 1: Load and merge data ---
# Load the main metadata sheet from mmc1.xlsx which contains patient 'Age'.
meta_df = pd.read_excel(file_path_meta, sheet_name='UCEC_CPTAC3_meta_table_V2.1',
index_col=0)

# Load the APM subtypes sheet from mmc7.xlsx which contains the 'APP_Z_score'.
apm_df = pd.read_excel(file_path_apm, sheet_name='B-APM subtypes', index_col=0)

# Select only the necessary columns
patient_info_df = meta_df[['Age']]
apm_scores_df = apm_df[['APP_Z_score']]

# Merge the two dataframes based on the sample ID index
merged_df = patient_info_df.join(apm_scores_df, how='inner')
subtask1_answer = "Data loaded and merged successfully. The merged data contains 'Age' and
'APP_Z_score' for each patient."

# --- Solution for Subtask 2: Clean the data ---
# Convert columns to numeric, coercing errors to NaN
merged_df['Age'] = pd.to_numeric(merged_df['Age'], errors='coerce')
merged_df['APP_Z_score'] = pd.to_numeric(merged_df['APP_Z_score'], errors='coerce')

# Drop rows with missing values in 'Age' or 'APP_Z_score'
cleaned_df = merged_df.dropna(subset=['Age', 'APP_Z_score'])
subtask2_answer = "Data cleaned. Rows with missing 'Age' or 'APP_Z_score' have been removed."

# --- Solution for Main Task: Find the age of the patient with the lowest APP-Z score ---
# Find the index of the patient with the lowest APP-Z score
patient_with_lowest_score_idx = cleaned_df['APP_Z_score'].idxmin()

# Get the age of that patient
age_of_patient = cleaned_df.loc[patient_with_lowest_score_idx, 'Age']

# The final answer is the age of the patient, converted to an integer.
final_answer = int(age_of_patient)

# Print the final results in JSON format
print(json.dumps(
    {
        "subtask-1": subtask1_answer,
        "subtask-2": subtask2_answer,
        "main-task": final_answer
    },
    indent=4
))
```

Master-Slave — Generated Response: **74**

```
import pandas as pd
import json

def get_age_and_app_scores(files):
    """
    Loads patient age and APP protein abundance scores from the supplementary files.

    Args:
        files (list): A list of all file paths in the dataset.

    Returns:
        pandas.DataFrame: A DataFrame with 'Age' and 'APP_log2_abundance' for each sample.
    """
    # Find the metadata and proteomics files from the list
    meta_file = next((f for f in files if 'mmc1.xlsx' in f), None)
    prot_file = next((f for f in files if 'mmc2.xlsx' in f), None)

    if not meta_file or not prot_file:
        raise FileNotFoundError("Required metadata (mmc1) or proteomics (mmc2) file not found.")

    # Load the clinical metadata to get patient age
    df_meta = pd.read_excel(meta_file, sheet_name='UCEC_CPTAC3_meta_table_V2.1', index_col=0)
    age_data = df_meta[['Age']]

    # Load the global proteomics data to get protein scores
    df_prot = pd.read_excel(prot_file, sheet_name='A-global-proteomics', index_col=0)

    # Check if the 'APP' protein is in the proteomics data
    if 'APP' in df_prot.index:
        # Extract the row for 'APP', transpose it to a column, and rename it
        app_scores = df_prot.loc[['APP']].T
        app_scores.rename(columns={'APP': 'APP_log2_abundance'}, inplace=True)
    else:
        # If APP is not found, create a DataFrame with NaNs to allow merging
        app_scores = pd.DataFrame(index=df_meta.index, columns=['APP_log2_abundance'])

    # Combine the age and APP score dataframes
    combined_df = age_data.join(app_scores)

    # Remove samples that are missing either Age or the APP score
    combined_df.dropna(subset=['Age', 'APP_log2_abundance'], inplace=True)

    return combined_df

# List of all available files
file_list = [
    '1-s2.0-S0092867420301070-mmc1.xlsx',
    '1-s2.0-S0092867420301070-mmc2.xlsx',
    '1-s2.0-S0092867420301070-mmc3.xlsx',
    '1-s2.0-S0092867420301070-mmc4.xlsx',
    '1-s2.0-S0092867420301070-mmc5.xlsx',
    '1-s2.0-S0092867420301070-mmc6.xlsx',
    '1-s2.0-S0092867420301070-mmc7.xlsx'
]

# Execute the function to get the data
age_app_data = get_age_and_app_scores(file_list)

# Find the patient with the lowest APP-Z score
patient_with_lowest_app = age_app_data.loc[age_app_data['APP_log2_abundance'].idxmin()]

# Get the age of that patient
age = patient_with_lowest_app['Age']

# Create the final answer dictionary
final_answer = {
    "main-task": age
}

# Print the final answer in JSON format
print(json.dumps(final_answer, indent=4))
```

Figure 15: An example of the generated program by the blackboard system and master-slave system. The green box highlights where the blackboard system correctly selected the relevant files from the data lake, while the red box indicates where the master–slave system made an incorrect selection.

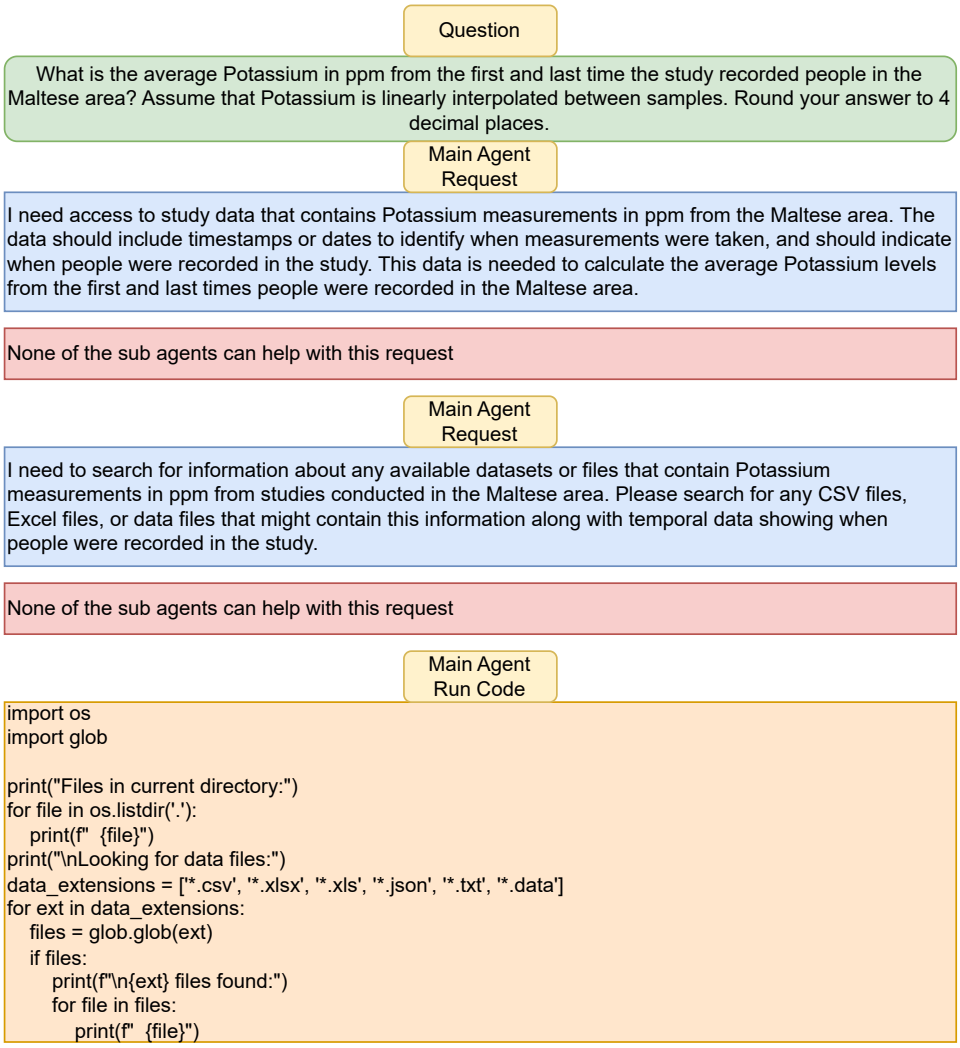


Figure 16: An example of the requesting operation that none of the sub-agents can help with this request. In this case, the main agent retries multiple times with different reformulations of the request, and upon repeated failure, resorts to alternative actions, such as executing code to locate the relevant files autonomously.

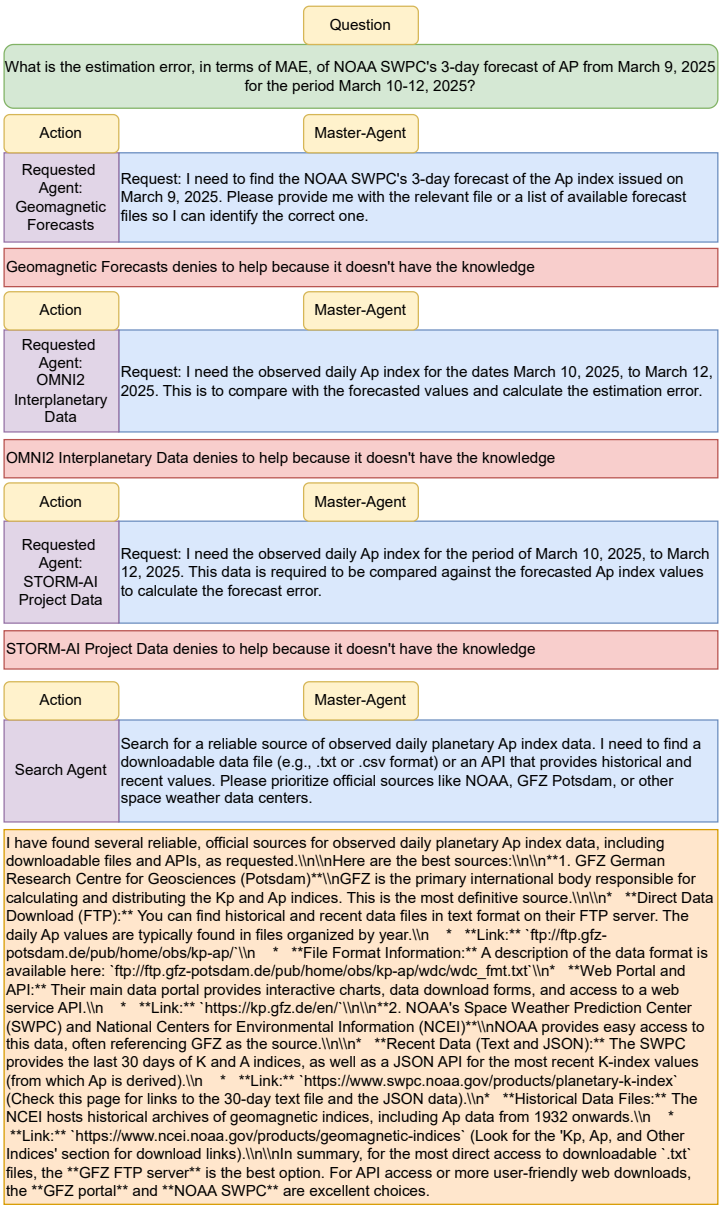


Figure 17: Example where the main controller in a Master–Slave system fails to route the query to the correct data sub-agent: after multiple retries, it falls back to the search agent, which returns only generic or incorrect information.

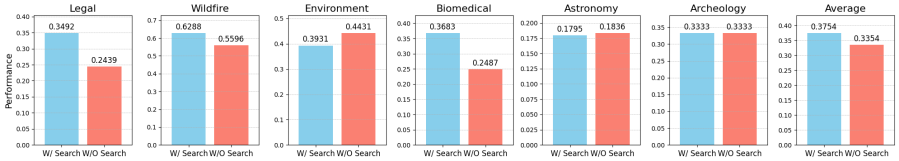


Figure 18: Performance of Blackboard System w/ and w/o search agent (Gemini 2.5 Pro).

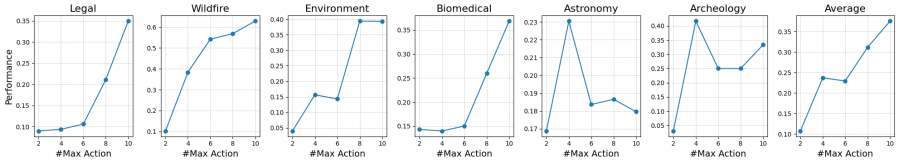


Figure 19: Performance of Blackboard System with various maximum actions by the main agent.

F Further Results Analysis

Scalability Comparison between Blackboard and Master–Slave Systems: We report relative, rather than absolute, performance gains to ensure fair comparison across datasets with varying task difficulties and score ranges. Absolute improvements can be misleading when the baseline performance or achievable score range differs substantially—for instance, a 5-point gain might be minor in an easy task with high baseline scores but significant in a challenging one. Relative gain normalizes these differences by measuring proportional improvement with respect to the baseline, enabling consistent comparison across heterogeneous tasks. Using this normalized metric, we analyze the scalability of the Blackboard architecture compared to the Master–Slave baseline across datasets with different data lake sizes (Figure 20(A)). Each point corresponds to a distinct task domain, and a fitted regression line reveals a clear positive correlation between data lake size and relative performance gain. This indicates that the Blackboard system scales more effectively as data environments grow larger, whereas the Master–Slave system exhibits limited scalability under such conditions.

Since each point in this analysis corresponds to a different benchmark or domain, the effect of data lake size is not fully isolated from other factors, such as the domain and difficulty of the tasks. To isolate it, we conduct a controlled experiment on the two largest data lakes, DA-Code and the Astronomy domain of KramaBench: we keep the tasks and their ground-truth files fixed and vary the number of distractor files to construct data lakes containing 33%, 67%, and 100% of the full data lake, and evaluate both systems using Gemini 2.5 Pro as the LLM. As shown in Figure 20(B), the relative gain of the Blackboard system over the Master–Slave system increases with data lake size on both benchmarks, from 15% to 49% and 70% on DA-Code, and from 34% to 74% and 112% on Astronomy. Since the tasks are held fixed, this provides stronger evidence that the advantage of the Blackboard architecture grows as data discovery becomes more difficult.

Runtime and Cost Analysis: To assess the efficiency–cost trade-off of the Blackboard system relative to the RAG and Master–Slave baselines, we randomly sampled 50 questions from the KramaBench benchmark spanning all domains and measured both runtime and cost per question for each method. Unlike the RAG and Master–Slave baselines, which execute their component calls sequentially following the ReAct framework, the Blackboard architecture parallelizes sub-agent interactions: once the main agent posts a request to the shared blackboard, the corresponding sub-agents process it independently. As shown in Figure 21, the runtime of all three systems lies in a narrow band (132.0–145.2 seconds), indicating no significant difference in latency. In terms of monetary cost, the Blackboard system is more expensive per question (approximately $2.3\times$ the cost of RAG and $1.8\times$ that of Master–Slave), reflecting its increased token usage. However, this additional cost translates into substantial performance gains—54.1% over RAG and 18.8% over Master–Slave—so that Blackboard delivers markedly better answer quality while maintaining comparable runtime, offering a favorable accuracy–cost trade-off.

Since the Blackboard system incurs a higher inference cost, a natural question is whether its performance gains can be attributed solely to this larger computation budget. To study this, we conduct a budget-matched experiment in which the Master–Slave system is given the same inference budget as the Blackboard system. Specifically, we allow the main agent of the Master–Slave system to continue beyond its original limit of $T = 10$ actions until its inference cost per question reaches the

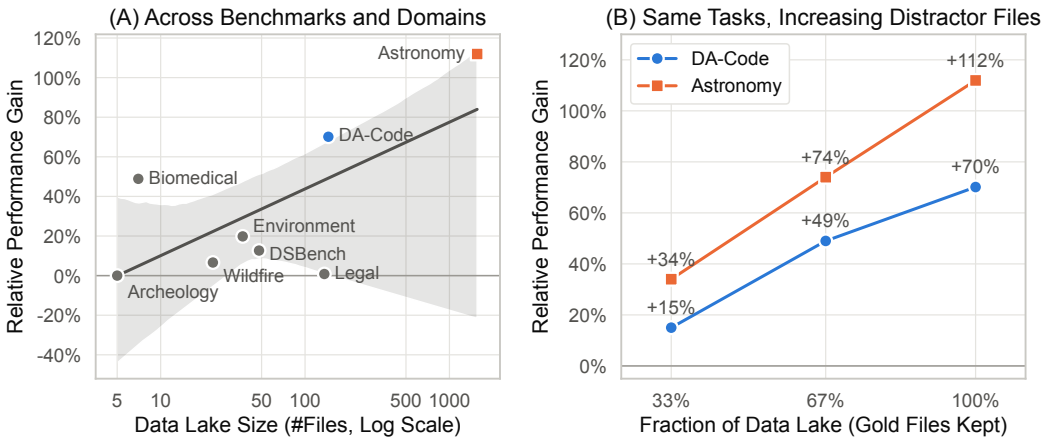


Figure 20: Relative performance gain of the Blackboard system over the Master–Slave system as a function of data lake size, using Gemini 2.5 Pro as the LLM. **(A)** Relative gain across benchmarks and KramaBench domains with different data lake sizes (log scale); each point corresponds to a task domain, the line represents the fitted regression line, and the shaded area shows the confidence interval. **(B)** Controlled experiment on DA-Code and the Astronomy domain of KramaBench, where the tasks and their ground-truth files are kept fixed while the number of distractor files is varied to construct data lakes containing 33%, 67%, and 100% of the full data lake. In both settings, the relative gain increases with the size of the data lake, suggesting that the Blackboard architecture scales more effectively with larger data lakes.

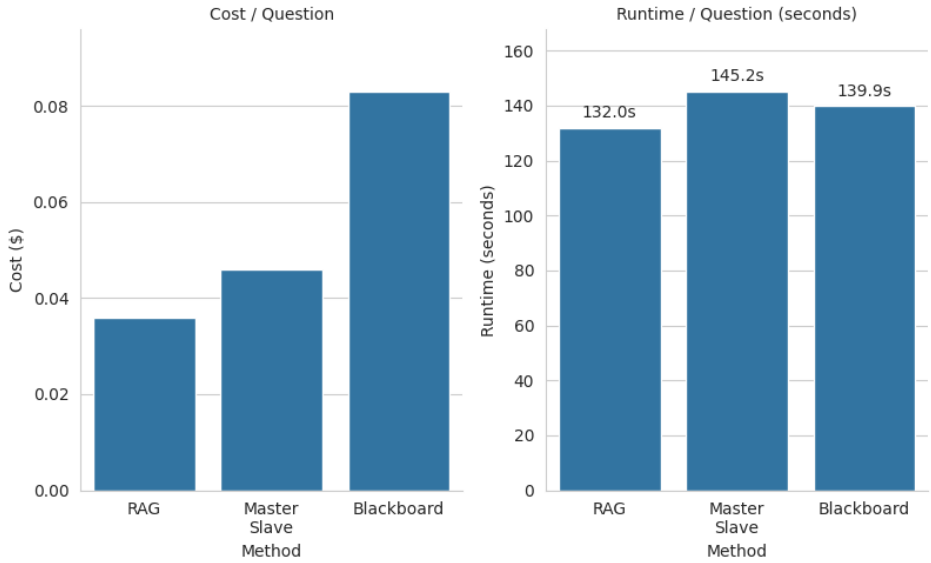


Figure 21: Runtime and cost analysis of the RAG, Master–Slave, and Blackboard systems on 50 examples from the KramaBench benchmark. We use Gemini 2.5 Pro as the LLM.

Table 4: Budget-matched comparison between the Blackboard and Master-Slave systems using Gemini 2.5 Pro as the LLM. In the budget-matched setting, the main agent of the Master-Slave system is allowed to continue beyond its original limit of $T = 10$ actions until its average inference cost per question matches that of the Blackboard system (Figure 21). Results for Master-Slave and Blackboard with $T = 10$ are taken from Table 1. The best results are highlighted in **bold**.

Method	KramaBench (Avg.)	DSBench	DA-Code	Avg.
Master-Slave ($T = 10$)	32.16%	34.38%	5.49%	24.01%
Master-Slave (budget-matched)	33.23%	35.57%	7.14%	25.31%
Blackboard ($T = 10$)	**37.53%**	**38.73%**	**9.34%**	**28.53%**

average cost per question of the Blackboard system (Figure 21). This provides the Master–Slave system with additional opportunities for planning, reasoning, code execution, and invoking helper agents, while controlling for the overall computation budget. The results of this experiment, using Gemini 2.5 Pro as the LLM, are reported in Table 4. As shown, the additional budget improves the performance of the Master–Slave system only slightly (from 24.01% to 25.31% on average), and the Blackboard system remains better on all three benchmarks: 37.53% versus 33.23% on KramaBench, 38.73% versus 35.57% on DSBench, and 9.34% versus 7.14% on DA-Code. This indicates that the performance gains of the Blackboard system are not attributable solely to its higher token usage or to additional reasoning opportunities; instead, the way this budget is used—broadcasting requests and allowing agents with direct knowledge of their local data to self-select—provides benefits beyond simply allocating more computation to a centralized controller. Nevertheless, we characterize this as an accuracy–cost trade-off rather than an unconditional advantage: in highly cost-sensitive settings, RAG or the Master–Slave system may be preferable, whereas the Blackboard system is more suitable when answer quality is the primary objective. The cost of the Blackboard system can be further reduced, for example, by using confidence thresholds, restricting the number of participating agents, or employing smaller LLMs for assessing the relevance of helper agents to each request.

The analysis above focuses on the online question-answering phase. In addition, both the Blackboard and Master–Slave systems require an identical offline phase, in which the data lake is clustered and each file agent analyzes its assigned files; therefore, this phase does not explain the performance difference between them. Using Gemini 2.5 Pro, the offline analysis takes 243.6 seconds and costs $2.40 on average per file agent, each handling multiple data files, resulting in a total cost between $4.2 and $42.1 per dataset depending on the number of file agents. Since file agents analyze their assigned clusters independently, these analyses can be executed in parallel; moreover, they are performed only once and amortized across all subsequent questions over the same data lake. For frequently changing data lakes, only newly added or affected file agents need to be reanalyzed rather than rebuilding the entire system.

Analysis of Helper Agents' Participation: To better understand why the Blackboard system outperforms the Master–Slave system, we analyze how helper agents participate in problem solving using Gemini 2.5 Pro as the LLM. On average, the main agent of the Master–Slave system invokes helper agents 3.8 times per problem, including retries after selecting agents that do not hold the required data (e.g., Figure 17 in Appendix E), whereas the main agent of the Blackboard system posts only 2.2 requests per problem, each receiving responses from only 1.2 helper agents on average. Moreover, 79% of the volunteering file agents are responsible for at least one ground-truth file, showing that helper agents can reliably assess their own relevance to a request. Thus, the higher cost of the Blackboard system mainly arises from helper agents independently evaluating each broadcast request, rather than from returning more responses to the main agent.

Compare with Other More Advanced Data Science Baselines: To further evaluate our method against advanced data-science-oriented agent frameworks, we compare the Blackboard system with Data Interpreter [Hong et al., 2025] and AutoGen [Wu et al., 2023], both designed for multi-agent data analysis and reasoning. The results on the KramaBench benchmark are presented in Table 5. As shown, the Blackboard system outperforms these baselines in five out of six tasks as well as on the overall average, highlighting its stronger adaptability and coordination capabilities across diverse data-science scenarios.

Table 5: Results on the KramaBench benchmark using Gemini 2.5 Pro as the underlying LLM.

Method	Archaeology	Astronomy	Biomedical	Environment	Legal	Wildfire	Average
Data Interpreter	**41.67**%	12.72%	28.05%	9.87%	30.04%	59.67%	30.34%
AutoGen	16.67%	4.39%	7.25%	19.38%	26.38%	41.76%	19.30%
Blackboard	33.33%	**17.95**%	**36.83**%	**39.31**%	**34.92**%	**62.88**%	**37.53**%

Table 6: Results of the Blackboard and Master-Slave systems on the benchmarks using Gemini 3.1 Pro as the LLM. The best results are highlighted in **bold**. The KramaBench categories are abbreviated: Arc. (Archaeology), Ast. (Astronomy), Bio. (Biomedical), Env. (Environment), Leg. (Legal), and Wild. (Wildfire).

Method	**KramaBench**							**DS Bench**	**DA-Code**	**Avg.**
	Arc.	Ast.	Bio.	Env.	Leg.	Wild.	Avg.			
Master-Slave	33.33%	21.32%	32.28%	59.16%	54.58%	58.23%	43.15%	43.47%	8.79%	31.80%
Blackboard	**50.00%**	**34.36%**	**45.31%**	**61.17%**	**61.25%**	**61.18%**	**52.21%**	**47.43%**	**13.73%**	**37.79%**

Results with a More Recent LLM: To examine whether the advantage of the Blackboard architecture persists with more recent frontier LLMs, we additionally evaluate the Blackboard and Master–Slave systems using Gemini 3.1 Pro as the backbone LLM. The results are reported in Table 6. As shown, the Blackboard system outperforms the Master–Slave system on all three benchmarks—52.21% versus 43.15% on KramaBench, 47.43% versus 43.47% on DSBench, and 13.73% versus 8.79% on DA-Code—as well as in all six domains of KramaBench. This indicates that the improvement stems from the proposed communication architecture rather than from the choice of a particular backbone LLM, and that it persists as the underlying LLMs become more capable.

Effect of Clustering Based on File Name and Content and Number of Clusters on the Performance: As described in §3, we employ Gemini 2.5 Pro to perform file clustering based on filenames. However, given that file contents can be lengthy, it is impractical to provide the full text of each file to the model for clustering. To address this, we additionally experiment with content-based clustering using a semantic embedding model. Specifically, we encode each file's content using E5-Large [Wang et al., 2022] and apply the KMeans algorithm [Lloyd, 1982] to partition the files into multiple clusters. Unlike Gemini, which automatically determines the number of clusters, KMeans requires this value to be specified. We evaluate settings with 2, 4, 8, and 16 clusters and use datasets containing at least 100 files—namely, the Legal and Astronomy subsets from the KramaBench and DA-Code datasets.

The results of the content-based clustering are presented in Figure 22. Overall, increasing the number of clusters leads to higher performance, as it allows each sub-agent to specialize on a smaller subset of files and perform more targeted reasoning. Furthermore, to compare filename-based clustering with Gemini 2.5 Pro against content-based clustering using E5-Large embeddings and random clustering, we report the results in Figure 23. As shown, clustering based on semantic content yields consistently better performance than clustering by filename alone or random clustering. This indicates that our proposed framework generalizes effectively to embedding-based clustering and does not rely solely on large language models for grouping data files.

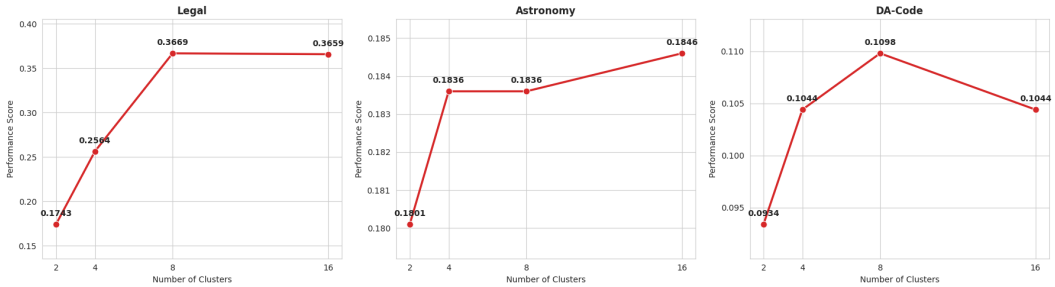


Figure 22: Effect of the number of clusters on the performance of the Blackboard system. Clustering is performed using the KMeans algorithm with E5-Large as the embedding model. We use Gemini 2.5 Pro as the LLM.

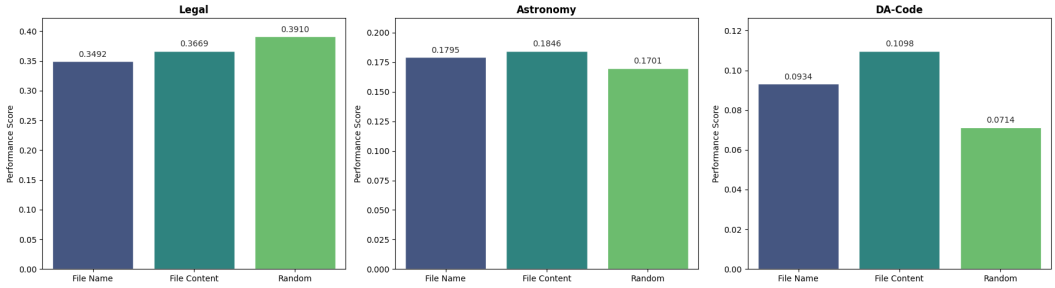


Figure 23: Effect of clustering based on filename using Gemini 2.5 Pro and based on file content using the KMeans algorithm with E5-Large as the embedding model. We use Gemini 2.5 Pro as the LLM.

G Example IDs for Filtered Datasets

We retained the following example IDs from the instances in the DSBench dataset:

- 00000001_question17
- 00000001_question7
- 00000001_question16
- 00000001_question13
- 00000001_question12
- 00000001_question9
- 00000001_question10
- 00000001_question11
- 00000001_question18
- 00000001_question15
- 00000004_question50
- 00000004_question45
- 00000004_question42
- 00000004_question48
- 00000004_question47
- 00000004_question49
- 00000004_question44
- 00000004_question41
- 00000004_question43
- 00000005_question21
- 00000005_question29
- 00000005_question28
- 00000005_question24
- 00000005_question25
- 00000005_question23
- 00000005_question26
- 00000005_question20
- 00000005_question27
- 00000006_question17
- 00000006_question22
- 00000006_question21
- 00000006_question18
- 00000006_question24
- 00000006_question25
- 00000006_question19
- 00000006_question23
- 00000006_question26
- 00000006_question20
- 00000007_question17
- 00000007_question22
- 00000007_question7
- 00000007_question16
- 00000007_question6
- 00000007_question5
- 00000007_question13
- 00000007_question12
- 00000007_question21
- 00000007_question9
- 00000007_question10
- 00000007_question11
- 00000007_question3
- 00000007_question18
- 00000007_question4
- 00000007_question19
- 00000007_question23
- 00000007_question14
- 00000007_question2
- 00000007_question15
- 00000007_question20
- 00000007_question1
- 00000007_question8
- 00000008_question33
- 00000008_question31
- 00000008_question28
- 00000008_question32
- 00000008_question34
- 00000010_question17
- 00000010_question7
- 00000010_question16
- 00000010_question6
- 00000010_question5
- 00000010_question13
- 00000010_question12
- 00000010_question9
- 00000010_question10

- 00000010_question11
- 00000010_question3
- 00000010_question18
- 00000010_question4
- 00000010_question19
- 00000010_question14
- 00000010_question2
- 00000010_question15
- 00000010_question20
- 00000010_question1
- 00000010_question8
- 00000011_question7
- 00000011_question6
- 00000011_question5
- 00000011_question3
- 00000011_question4
- 00000011_question2
- 00000011_question1
- 00000011_question8
- 00000012_question7
- 00000012_question6
- 00000012_question5
- 00000012_question9
- 00000012_question10
- 00000012_question3
- 00000012_question4
- 00000012_question2
- 00000012_question1
- 00000012_question8
- 00000013_question17
- 00000013_question22
- 00000013_question16
- 00000013_question21
- 00000013_question18
- 00000013_question24
- 00000013_question19
- 00000013_question23
- 00000013_question20
- 00000016_question7
- 00000016_question6
- 00000016_question5
- 00000016_question12
- 00000016_question9
- 00000016_question11
- 00000016_question3
- 00000016_question2
- 00000016_question1
- 00000016_question8
- 00000017_question39
- 00000017_question33
- 00000017_question38
- 00000017_question29
- 00000017_question37
- 00000017_question30
- 00000017_question32
- 00000017_question35
- 00000017_question34
- 00000017_question36
- 00000018_question24
- 00000018_question25
- 00000018_question23
- 00000019_question6
- 00000019_question13
- 00000019_question9
- 00000019_question10
- 00000019_question14
- 00000019_question15
- 00000020_question33
- 00000020_question31
- 00000020_question29
- 00000020_question37
- 00000020_question30
- 00000020_question28
- 00000020_question32
- 00000020_question35
- 00000020_question34
- 00000020_question36
- 00000022_question39
- 00000022_question33
- 00000022_question38
- 00000022_question40
- 00000022_question37
- 00000022_question35
- 00000022_question34
- 00000022_question36
- 00000025_question39
- 00000025_question33
- 00000025_question38
- 00000025_question40
- 00000025_question37
- 00000025_question35
- 00000025_question41
- 00000025_question34
- 00000025_question36
- 00000027_question13
- 00000027_question12
- 00000027_question14
- 00000027_question15
- 00000029_question7
- 00000029_question6
- 00000029_question5
- 00000029_question9
- 00000029_question10
- 00000029_question3
- 00000029_question4
- 00000029_question2
- 00000029_question1
- 00000029_question8
- 00000030_question7
- 00000030_question6
- 00000030_question5
- 00000030_question3
- 00000030_question4
- 00000030_question2
- 00000030_question1
- 00000032_question7
- 00000032_question6
- 00000032_question5
- 00000032_question3
- 00000032_question4
- 00000032_question2
- 00000032_question1
- 00000033_question7
- 00000033_question6
- 00000033_question3
- 00000033_question4
- 00000033_question2
- 00000033_question1
- 00000033_question8
- 00000034_question7
- 00000034_question16
- 00000034_question6
- 00000034_question5
- 00000034_question13
- 00000034_question12
- 00000034_question9
- 00000034_question10
- 00000034_question11
- 00000034_question3
- 00000034_question4
- 00000034_question14
- 00000034_question2
- 00000034_question15
- 00000034_question1
- 00000034_question8
- 00000035_question7
- 00000035_question6
- 00000035_question5
- 00000035_question9
- 00000035_question3
- 00000035_question4
- 00000035_question2
- 00000035_question1
- 00000035_question8
- 00000038_question5
- 00000038_question3
- 00000038_question2
- 00000038_question1
- 00000043_question17
- 00000043_question7
- 00000043_question16
- 00000043_question6
- 00000043_question5
- 00000043_question13
- 00000043_question12
- 00000043_question9
- 00000043_question10
- 00000043_question11
- 00000043_question3
- 00000043_question18
- 00000043_question4
- 00000043_question19
- 00000043_question14
- 00000043_question2
- 00000043_question15
- 00000043_question20
- 00000043_question1
- 00000043_question8

We retained the following example IDs from the instances in the DA-Code dataset:

- di-text-001
- di-text-002
- di-text-003
- di-text-004
- di-text-005
- di-text-006
- di-text-007
- di-text-008
- di-text-009
- di-text-010
- di-text-011
- di-text-012
- di-text-013
- di-text-014
- di-text-015
- di-text-016
- di-text-017
- di-text-018
- di-text-019
- di-text-020
- di-text-021
- di-text-023
- di-text-024
- di-text-025
- di-text-027
- di-text-028
- di-text-029
- di-text-030
- di-text-031
- di-text-032
- di-text-033
- di-text-034
- di-text-035
- di-text-036
- di-text-037
- di-text-038
- di-text-039
- di-text-040
- di-text-041
- di-csv-001
- di-csv-002
- di-csv-003
- di-csv-005
- di-csv-006
- di-csv-007
- di-csv-008
- di-csv-009
- di-csv-010
- di-csv-011
- di-csv-012
- di-csv-013
- di-csv-014
- di-csv-015
- di-csv-016
- di-csv-017
- di-csv-018
- di-csv-019
- di-csv-020
- di-csv-021
- di-csv-022
- di-csv-023
- dm-csv-001
- dm-csv-002
- dm-csv-009
- dm-csv-010
- dm-csv-011
- dm-csv-016
- dm-csv-019
- dm-csv-020
- dm-csv-021
- dm-csv-028
- dm-csv-029
- dm-csv-032
- dm-csv-034
- dm-csv-038
- dm-csv-041
- dm-csv-049
- dm-csv-050
- dm-csv-055
- data-sa-011
- data-sa-012
- data-sa-022
- data-sa-034
- data-sa-035
- data-sa-037
- data-sa-041
- data-sa-044
- data-sa-047
- data-sa-065
- data-sa-067
- data-sa-068

H Limitations

Despite the strong empirical performance of the proposed blackboard multi-agent framework, several limitations remain. First, the framework incurs higher computational cost compared to simpler baselines, primarily due to increased token usage and parallel interactions among multiple agents. While this additional cost is justified by improved performance, it may limit applicability in resource-constrained environments. However, despite the increased token consumption, the overall runtime remains comparable to baseline methods due to parallelization and efficient coordination among agents.

Second, the effectiveness of the system depends on the quality, coverage, and diversity of helper agents. In scenarios where agents have incomplete, overlapping, or noisy knowledge, the self-selection mechanism may result in missing, redundant, or conflicting contributions, which can negatively impact downstream reasoning. Moreover, the current framework assumes relatively static agent capabilities and does not explicitly address dynamic environments where data distributions or agent expertise evolve over time. While we do not explicitly study such dynamic settings in this work, the underlying blackboard design naturally supports extensibility: since requests are broadcast and agents autonomously decide whether to participate, new agents or data sources can be seamlessly added without modifying the coordination mechanism, allowing the system to scale and adapt as capabilities evolve.

Finally, our evaluation is limited to data science benchmarks with structured tasks and executable ground-truth programs. Although these benchmarks capture key aspects of data discovery, they do not cover the broader range of applications where such a framework could be beneficial. While we focus on data science in this work, the proposed blackboard architecture is general and can be readily applied to other domains that require distributed reasoning, coordination, or information aggregation (e.g., software engineering, scientific discovery, or complex decision-making tasks). Exploring and validating its effectiveness in these broader settings remains an important direction for future work.

I Impact Statement

This paper introduces a novel multi-agent communication paradigm designed to address the data discovery bottleneck, which currently limits the practical application of large language models (LLMs) in real-world data science. By enabling autonomous agents to more effectively locate and interpret relevant information within massive, heterogeneous data lakes, this work has the potential to accelerate data-driven insights across critical domains such as biomedical research, environmental monitoring, and legal analysis. Furthermore, by replacing rigid centralized control with a flexible, decentralized blackboard architecture, this research promotes the development of more scalable and autonomous AI systems. While the advancement of autonomous agents carries broader societal implications regarding workforce transitions and the energy consumption of large-scale models, this work focuses on improving the reliability and efficiency of scientific workflows.

J Large Language Model Usage for Writing

In this paper, we employ LLMs—specifically Gemini and ChatGPT—as general-purpose writing tools. Draft text is provided to these models, which are then asked to improve the writing by correcting grammatical errors and refining the structure. The edited text is then verified and edited if needed. The use of LLMs in this paper is limited strictly to text refinement. They were not employed for tasks such as generating any new content or references.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly summarize the core contributions and align with the presented methods and results, without overstating the scope. They accurately reflect both the proposed approach and the empirical findings discussed in the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: It is provided in Appendix H.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides sufficient details about the experimental setup, including datasets, model configurations, and evaluation protocols in Section 4.1, enabling others to reproduce the main results. Key design choices and implementation specifics are clearly described to support replication.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The code cannot be released due to proprietary constraints; however, the underlying datasets used in the experiments are publicly available. The paper provides sufficient methodological details to enable reproduction of the main results using these public resources.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper clearly specifies the experimental setup, including data splits, model configurations, and key hyperparameters in Section 4.1. It also describes how hyperparameters were selected, allowing readers to fully understand and interpret the reported results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Experiments were run once per setting and no error bars are reported. We use a deterministic or near-deterministic decoding configuration (low temperature of 0.1), which reduces randomness, but we do not report variability across multiple runs or seeds due to high computational costs.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports the main compute resources used for evaluation, including the type of GPUs and general runtime requirements for experiments. This information is sufficient to give a clear understanding of the computational budget needed to reproduce the reported results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research was conducted in accordance with the NeurIPS Code of Ethics, with no use of sensitive or prohibited data and no procedures that raise ethical concerns. The experiments rely on publicly available datasets and standard evaluation protocols, ensuring compliance with ethical research practices.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The impact statement is provided in Appendix I.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper properly credits all external datasets, codebases, and models used in the experiments, and cites their original sources. It also specifies the versions where applicable and respects the stated licenses and terms of use for each asset.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.