Stat

WILEY

SPECIAL ISSUE ARTICLE

VDSAgents: A PCS-Guided Multi-Agent System for Veridical Data Science Automation

Yunxuan Jiang[1] | Silan Hu[2] | Xiaoning Wang[3] | Yuanyuan Zhang[4] | Xiangyu Chang[1]

[1]School of Management, Xi'an Jiaotong University, Xi'an, China | [2]School of Computing, National University of Singapore, Queenstown, Singapore | [3]School of Data Science and Media Intelligence, Communication University of China, Beijing, China | [4]Beijing Baixingkefu Network Technology Co., Ltd, Beijing, China

Correspondence: Xiaoning Wang (sdwangxiaoning@cuc.edu.cn)

Received: 1 August 2025 | **Revised:** 28 October 2025 | **Accepted:** 17 November 2025

ABSTRACT

Large language models (LLMs) become increasingly integrated into data science workflows for automated system design. However, these LLM-driven data science systems rely solely on the internal reasoning of LLMs, lacking guidance from scientific and theoretical principles. This limits their trustworthiness and robustness, especially when dealing with noisy and complex real-world datasets. This paper provides `VDSAgents`[1], a multi-agent system grounded in the predictability–computability–stability (PCS) principles (Yu and Kumbier, 2020). proposed in the veridical data science (VDS) (Yu and Barter, 2024). Guided by PCS principles, the system implements a modular workflow for data cleaning, feature engineering, modeling and evaluation. Each phase is handled by an elegant agent, incorporating perturbation analysis, unit testing and model validation to ensure both functionality and scientific auditability. We evaluate `VDSAgents` on nine datasets with diverse characteristics, comparing it with state-of-the-art end-to-end data science systems, such as `AutoKaggle` and `DataInterpreter`, using DeepSeek-V3 and GPT-4o as backends. `VDSAgents` consistently outperforms the results of `AutoKaggle` and `DataInterpreter`, which validates the feasibility of embedding PCS principles into LLM-driven data science automation.

1 | Introduction

Data science has emerged as a multidisciplinary field that integrates statistics, computer science, mathematics and domain knowledge to extract meaningful insights and guide decision-making from complex data (Yu and Kumbier 2020). Its scope spans the entire data science lifecycle (DSLC), from collection and preprocessing to modeling, validation and knowledge refinement, and plays a vital role in decision-making in scientific, industrial and policy domains (Cao 2017; Provost and Fawcett 2013). A typical DSLC is illustrated in Figure 1 proposed by Yu and Barter (2024), which outlines the key stages of data science-based research. As data continue to grow in volume, complexity and heterogeneity, standard data science approaches are increasingly insufficient to meet the demands for trustworthiness and robustness. This has fueled the development of automated and principled DSLC.

Recent advances in large language models (LLMs), especially frontier models like GPT-4 (Achiam et al. 2023) and DeepSeek-V3 (DeepSeek-AI 2024), have significantly reshaped the landscape of data science automation. Using prompt engineering, tool integration and code generation capabilities, LLMs have been incorporated into systems that perform various stages of the data science pipeline (Li et al. 2024; Sun et al. 2024; Hong et al. 2024). A new paradigm has emerged, the agent-based approach, in which LLMs are organized into structured, role-based entities capable of simulating data scientists in end-to-end workflows. As Tu et al. (2024). observe, LLMs are increasingly positioned as strategic collaborators, shifting practitioners from manual operations to high-level planning and oversight.

Despite the promise of this approach, current agent-based data science systems face several persistent challenges. Existing frameworks like `AutoKaggle` (Li et al. 2024), `LAMBDA` (Sun

© 2026 John Wiley & Sons Ltd.

et al. 2024) and `DataInterpreter` (Hong et al. 2024) depend primarily on the intrinsic reasoning ability of the LLM to plan and execute multistep tasks. However, this autonomy often results in brittle execution paths, low reproducibility and limited robustness, especially in the presence of noisy, missing or structurally inconsistent data. These systems typically lack principled mechanisms to guide workflow design, evaluate stability or explore alternative analytic decisions. Consequently, while they are capable of producing seemingly valid pipelines, they often fail to ensure trustworthiness or consistency across datasets and contexts.

The veridical data science (VDS) framework (Yu and Kumbier 2020) provides a theory-based foundation for addressing these issues. Based on predictability–computability–stability (PCS) principles, VDS advocates data science as a process of critical and transparent reasoning rather than a mere algorithm execution. It emphasizes the importance of systematically examining model choices, testing stability via perturbation and ensuring analytic reproducibility. In this work, we propose integrating VDS into the architecture of LLM-agent systems by utilizing it as a structured external planning framework. This work leverages PCS principles and proposes a novel agent-based data science framework, which is naturally referred to as `VDSAgents`. Rather than relying solely on LLMs to autonomously generate task plans, we provide a predefined multistage skeleton grounded in DSLC (see Figure 1). This structure divides the pipeline into phases, including problem formulation, data cleaning and exploration, feature engineering and modeling and result evaluation. Each stage is managed by a dedicated agent, and a central `PCS-Agent` operates across all phases to assess and improve the predictability, computability and stability of the overall workflow. The `PCS-Agent` offers theoretical feedback at all levels of the workflow—questioning data credibility, suggesting alternative problem framings and enforcing reproducibility checks. This design enables the system to systematically explore diverse analytic paths, explicitly model uncertainty and ensure more reliable results using PCS principles (Yu and Barter 2024).

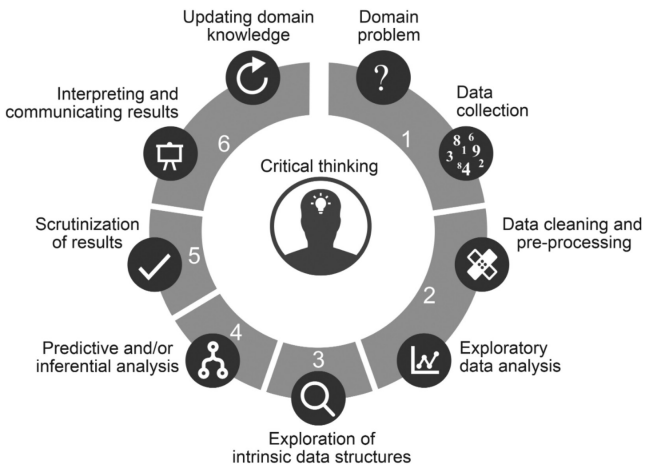


FIGURE 1 | Illustration of the DSLC. It includes six stages: (1) identifying and formulating domain problems and collecting data; (2) data cleaning, preprocessing and early exploration; (3) optional structural analysis and data mining; (4) optional modeling and statistical inference; (5) evaluation and validation of results and (6) interpretation, communication and domain knowledge update.

The main contributions of this paper are summarized as follows:

- **`VDSAgents` framework:** We propose the first multi-agent system that systematically integrates the DSLC into LLM-based architectures, guided by the PCS principles. The framework designs a dedicated `PCS-Agent` to guide all other agents in the DSLC.
- **Scientific tool integration:** A modular tool set is developed to support code execution, unit testing, fault diagnosis and image-to-text transformation, enhancing robustness and flexibility.
- **Paradigm advancement:** This work proposes a new paradigm of automation for trustworthy AI-assisted data science, bridging VDS and LLM agent methodologies.

We validate the proposed framework through systematic experiments on real-world datasets. Our results demonstrate that `VDSAgents` achieves superior robustness and predictive performance compared to representative LLM-driven systems.

2 | Related Work

2.1 | LLMs for Data Science

LLMs have demonstrated powerful capabilities in natural language understanding, reasoning and code generation (Achiam et al. 2023; DeepSeek-AI 2024). These advances have led to their growing use in data science, where LLMs can assist with tasks such as data cleaning, exploratory data analysis (EDA), feature engineering, modeling and automated report writing (Li et al. 2024; Hong et al. 2024; Sun et al. 2024; Jiang et al. 2025). Their ability to follow natural language instructions enables users to perform complex analyses with minimal coding.

As LLMs become more embedded in data science workflows, the focus shifts from manual execution to oversight and validation (Tu et al. 2024). To support this transition, there is a pressing need to introduce external theoretical frameworks that ensure the transparency, stability and reproducibility of LLM-driven data analysis.

2.2 | VDS and PCS Principle

VDS is a principled framework proposed by Yu and Kumbier (2020). To promote robust, reliable and reproducible data science. It centres around the PCS principles: predictability, computability and stability, each addressing critical aspects of reliable data analysis (Yu 2013). Predictability ensures that models generalize to new data; computability emphasizes practical feasibility, and stability tests the sensitivity of results to data and decision perturbations.

Despite its growing influence on human-led workflows, the VDS framework has not yet been integrated into autonomous LLM-driven agent systems. We argue that PCS principles provide valuable guidance for managing uncertainty and enhancing reproducibility in LLM-driven pipelines. Our work makes

the first attempt to systematically apply PCS principles to guide agent behaviour across the entire DSLC.

2.3 | Multi-Agent Systems and Task Planning

Many LLM-based systems rely on internal planning methods such as ReAct (Yao et al. 2022) and Tree-of-Thoughts (ToT) (Yao et al. 2023) to structure reasoning and execution. However, these approaches often suffer from instability and lack of reproducibility in DSLC, where task dependencies are complex and results must be tightly controlled.

To mitigate these issues, recent frameworks have adopted multi-agent designs with explicit task decomposition. `AutoKaggle` (Li et al. 2024) structures the pipeline into dedicated agents for data pre-processing, modeling and evaluation, improving modularity and execution traceability. `LAMBDA` (Sun et al. 2024) similarly defines role-specific agents to coordinate modeling tasks. `DataInterpreter` (Hong et al. 2024) further enhances coordination with a hierarchical task graph that supports dynamic planning and revision at all stages.

These systems demonstrate that combining structured planning with agent-based collaboration can improve interpretability and robustness. Our framework extends this principle by integrating an external `PCS-Agent` as a critical thinker to guide multi-agent execution across the full DSLC.

2.4 | Tool Integration and Execution Reliability

Recent research shows that the integration of external mechanisms, such as unit testing, execution feedback and self-refinement, can significantly improve the reliability of LLM in complex tasks (Madaan et al. 2023). For example, `AutoKaggle` (Li et al. 2024) incorporates an executor with error capture capabilities that detects runtime failures and automatically triggers correction procedures, thus improving both task completion rates and execution stability. In addition, other studies emphasize the use of tool-enhanced pipelines for verification and debugging (Wang et al. 2023; Zhou et al. 2023). Beyond using predefined tools, recent approaches also allow LLMs to dynamically create, manage or adapt tools for specific tasks (Cai et al. 2023; Schick et al. 2023; Qian et al. 2023).

Unit testing has emerged as a practical approach to validating the logic of LLM-generated code (Zhou et al. 2023). Frameworks such as PAL (Gao et al. 2023) embed intermediate programmatic reasoning and use test cases to verify the correctness of intermediate steps. This helps ensure the internal consistency and verifiability of multi-step reasoning processes.

Our system integrates a modular and extensible toolset to support reliable execution. This includes components for code execution, fault detection, self-debugging, OCR-based image-to-text conversion and unit testing. All tools are designed to be dynamically callable by agents and are decoupled from specific back-end models, enabling flexible deployment across different environments.

In summary, LLM-driven data science systems are evolving toward greater structure, moving from single-model reasoning to multi-agent collaboration and tool-assisted execution. Although these systems have improved efficiency and task coverage, they still struggle with the complexity of real-world data, reasoning stability and the trustworthiness of results. Existing solutions, such as unit testing and workflow supervision, offer partial improvements but often lack a theoretical foundation. To address these challenges, we propose `VDSAgents`, a multi-agent framework guided by the PCS principles, aiming to support trustworthy, stable and reproducible automated DSLC.

3 | Methodology

3.1 | Overview of `VDSAgents`

In real-world data science scenarios, challenges such as complex task dependencies, inconsistent data quality and subjective modeling choices often compromise the trustworthiness, reproducibility and robustness of results. To address these issues, `VDSAgents` decomposes the workflow into the following five dedicated agents.

- `Define-Agent`: $\mathscr{A}_{\text{define}}$ – Formulates the problem and evaluates the data quality;
- `Explore-Agent`: $\mathscr{A}_{\text{explore}}$– Handles data cleaning, preprocessing and exploratory analysis;
- `Model-Agent`: $\mathscr{A}_{\text{model}}$– Conducts feature engineering, model training and prediction;
- `Evaluate-Agent`: $\mathscr{A}_{\text{evaluate}}$– Assesses model performance and interprets results;
- `PCS-Agent`: $\mathscr{A}_{\text{PCS}}$– Operates across all stages, enforcing predictability, computability and stability through perturbation analysis and reproducibility checks.

Figure 2 illustrates the high-level system architecture. The system has two key design components:

- **PCS-Guided Workflow.** The data science process is divided into five sequential stages: problem definition and evaluation of data quality, data cleaning and EDA, predictive modeling, evaluation of results and PCS-Guided perturbation and comparison. This structure ensures that each step is scientifically grounded and aligned with the PCS principles.
- **Modular Multi-Agent Architecture.** Each agent is responsible for executing a specific phase of the workflow and operates using *statements from the VDS book (Yu and Barter 2024) as prompts* and shared memory. The `PCS-Agent` continuously analyses intermediate outputs, performs perturbation testing and evaluates the stability and consistency of the results.

To further formalize the execution logic, we define a high-level algorithm (Algorithm 1) that governs the interactions between agents and tool usage across stages. Let $\Phi = \{\phi_1, \phi_2, \phi_3, \phi_4\}$

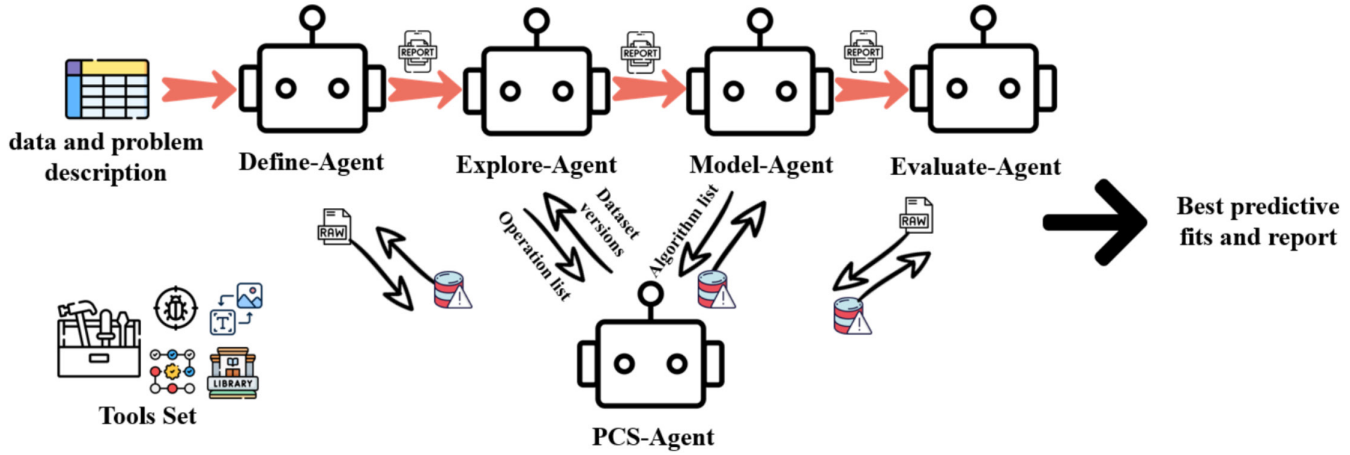


FIGURE 2 | Workflow architecture of the `VDSAgents`. Note that the `PCS-Agent` interacts with all stage-specific agents to evaluate predictability, computability and stability.

Algorithm 1: `VDSAgents` workflow

Input: Raw dataset D_{raw}

Output: Structured analysis report $\mathcal{R}$ and predictions $\hat{Y}_{\text{test}}$

Initialize system state $s_0 \leftarrow$ task description and problem definition;
Define stage sequence $\Phi = \{\phi_1, \phi_2, \phi_3, \phi_4\}$;
Define agent set $\mathcal{A} = \{\mathcal{A}_{\text{define}}, \mathcal{A}_{\text{explore}}, \mathcal{A}_{\text{model}}, \mathcal{A}_{\text{evaluate}}, \mathcal{A}_{\text{PCS}}\}$;
foreach *stage* $\phi \in \Phi$ **do**
  if $\phi = \phi_1$ **then**
    $\mathcal{T}_1 = \{\text{code executor, debug tool}\}$;
    $r_1 \leftarrow \mathcal{A}_{\text{define}}.\text{execute}(D_{\text{raw}}, \mathcal{T}_1)$;
    $\mathcal{A}_{\text{PCS}}.\text{analyze}(r_1, \mathcal{T}_1)$;
  else if $\phi = \phi_2$ **then**
    $\mathcal{T}_2 = \{\text{code executor, debug tool, ML tools, image-to-text}\}$;
    $D_{\text{clean}} \leftarrow \mathcal{A}_{\text{explore}}.\text{clean}(D_{\text{raw}}, \mathcal{T}_2)$;
    while $\neg$*unitTestsPassed*(D_{clean}) **do**
      $\mathcal{A}_{\text{explore}}.\text{delete}(D_{\text{clean}})$
      ;
    $D_{\text{clean}} \leftarrow \mathcal{A}_{\text{explore}}.\text{clean}(D_{\text{raw}}, \mathcal{T}_2)$;
  $E \leftarrow \mathcal{A}_{\text{explore}}.\text{EDA}(D_{\text{clean}}, \mathcal{T}_2)$;
  $D_1, \ldots, D_k \leftarrow \mathcal{A}_{\text{PCS}}.\text{perturb}(D_{\text{clean}}, k)$;
else if $\phi = \phi_3$ **then**
  $\mathcal{T}_3 = \{\text{code executor, debug tool, ML tools}\}$;
  $\mathcal{M}_{1:l} \leftarrow \mathcal{A}_{\text{model}}.\text{train}(D_{\text{clean}}, \mathcal{T}_3)$;
  results $\leftarrow \mathcal{A}_{\text{PCS}}.\text{reproduce}(D_{1:k}, \mathcal{M}_{1:l}, \mathcal{T}_3)$;
  $\mathcal{A}_{\text{PCS}}.\text{selectTopK}(\text{results})$;
  $\mathcal{A}_{\text{PCS}}.\text{generateReport}()$;
else
  $\mathcal{T}_4 = \{\text{code executor, debug tool}\}$;
  $\hat{Y}_{\text{test}} \leftarrow \mathcal{A}_{\text{evaluate}}.\text{model}(D_{\text{test}}, \mathcal{M}_{\text{best}}, \mathcal{T}_4)$;
return $\mathcal{R}, \hat{Y}_{test}$;

represent the different stages of problem definition and evaluation of data quality, data cleaning and EDA, predictive modeling and evaluation of results, s_t the state of the system at time t, r_a the output of agent a, T the set of unit tests in the current stage, k the number of perturbed datasets generated for robustness analysis, l the number of candidate models trained and $\hat{Y}_{\text{test}}$ the predictions of the final model. Here, $\text{delete}(D_{\text{clean}})$ denotes the removal of both the intermediate cleaned dataset and the generated code that produced it, upon unit-test failure.

The toolset $\mathcal{T}_i$ includes task-specific utilities such as a converter code executor, debug tool an machine learning (ML) library and image-to-text, which allows for stage-specific automation and recovery.

3.2 | PCS-Guided Planning and Perturbation

A key innovation of the `VDSAgents` lies in embedding PCS-Guided planning in each agent through structured prompting and shared memory. This ensures that the agents operate not only reactively, but also in accordance with principled scientific reasoning.

Each agent receives two layers of prompt instructions:

- **System Message:** defines the agent's role, scope of action and associated PCS principle;
- **Task-Specific Message:** includes upstream outputs, stage-specific objectives, expected output formats (e.g., Python, Markdown, JSON) and relevant domain constraints.

Agents also maintain an intermediate memory state, allowing them to incorporate decisions made in earlier stages and reason within the context of the entire pipeline. This forms a cohesive planning skeleton aligned with the DSLC. Among all agents, the `PCS-Agent` serves as the central coordinator: It applies critical thinking grounded in the PCS principles to continuously guide and evaluate the outputs of $\mathcal{A}_{\text{define}}$, $\mathcal{A}_{\text{explore}}$, $\mathcal{A}_{\text{model}}$ and $\mathcal{A}_{\text{evaluate}}$, ensuring stability, interpretability and robustness throughout the workflow.

For example, when collaborating with the `Explore-Agent`, the `PCS-Agent` operationalizes PCS principles by generating multiple perturbed versions of the dataset $\{D_1, D_2, \ldots, D_k\}$ using different strategies (e.g., alternative imputation methods, outlier treatments and feature transformations). Each perturbed dataset undergoes unit testing to verify semantic and structural validity. The system trains a corresponding model $\mathcal{M}_i$, forming a set of predictive fits (the pairing of an algorithm and a particular cleaned/preprocessed training dataset used for training the algorithm (Yu and Barter 2024)) $\mathcal{F}_i = (D_i, \mathcal{M}_i)$. These fits are compared on the basis of generalization performance and PCS-guided diagnostics, allowing the system to identify and report the most robust and reliable models.

Detailed functional modules and implementation of the `PCS-Agent`, including hypothesis generation, stability analysis and visualization-based evaluations, are summarized in the Supporting Information Appendix 7. These functions provide the operational backbone for the critical thinking process described above, enabling systematic guidance and evaluation at all stages of the workflow.

3.3 | Tool Infrastructure and Execution Flow

To support stable, modular and reproducible execution in different stages of the pipeline, `VDSAgents` is equipped with an extensible tool infrastructure $\mathcal{T}$. These tools are available to all agents and serve key roles such as code execution, logic validation, error handling and perturbation control.

3.3.1 | PCS-Guided ML Function Library

The core of the tool infrastructure is a modular ML function library $\mathcal{T}_{\text{ML}}$, which supports data pre-processing, feature engineering and structured perturbation. It is used by $\mathcal{A}_{\text{explore}}$ for cleaning and exploratory analysis, by $\mathcal{A}_{\text{model}}$ for feature construction and model fitting and by $\mathcal{A}_{\text{PCS}}$ to generate perturbations.

Each function is implemented in a standalone format with explicit parameter interfaces and operation semantics. The LLM can call these functions through natural language prompts by referencing predefined descriptions injected into the system message. This design ensures consistent behaviour across different agents and promotes traceability and reproducibility in multi-agent execution. For a complete list of functions and their descriptions, see Supporting Information Appendix B.1.

3.3.2 | Unit Testing

The unit test is a validation mechanism designed to systematically examine datasets for structural integrity and logical correctness (Technology 2024). Its primary purpose is to ensure that the processed datasets remain logically consistent and free of errors introduced during data perturbation and preprocessing steps.

After data cleaning, `Explore-Agent` invokes a suite of unit tests $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$ to verify the structural and statistical validity of the dataset. These tests detect issues such as missing values, unprocessed data loss, or duplicates. Each test outputs a structured result $\langle \text{name}, \text{passed}, \text{message} \rangle$ to guide downstream execution or debugging. This mechanism reinforces computability and stability in early processing. See Supporting Information Appendix B.2 for test details.

3.3.3 | Code Execution and Debugging

To ensure reliable code execution, each agent-generated script is handled by a `code executor`. If execution fails or unit tests are not passed, the system invokes a `debug tool` that identifies errors, generates repair suggestions and returns the corrected code. This self-healing mechanism supports up to $N_{\max}$

retries (default value $N_{\max} = 3$). Upon repeated failures, it triggers human intervention or rolls back to the original planning state. This design ensures robustness and recoverability in complex workflows. A schematic of this mechanism is provided in the Supporting Information Appendix C.

3.3.4 | Image-to-Text Support

To enhance visual reasoning during EDA, `VDSAgents` incorporates an image-to-text module $\mathcal{T}_{\text{OCR}}$ that extracts structured textual descriptions from graphical outputs such as histograms, box plots and heatmaps.

Given an image input I, the module returns a set of textual elements:

$$\mathcal{T}_{\text{OCR}}(I) \to \mathcal{V}_{\text{text}} = \{v_1, v_2, \ldots, v_n\}.$$

These include titles, axis labels, statistical extremes, trends and outliers. The resulting set $\mathcal{V}_{\text{text}}$ is then used by downstream agents (`Explore-Agent`, `PCS-Agent`) to assist in logic validation, anomaly interpretation and stability assessment.

3.3.5 | System Extensibility and Modularity

`VDSAgents` is built with modularity and extensibility in mind. On the model side, it defines an abstract interface $\mathcal{M}_{\text{LLM}}$ that supports interchangeable use of various LLMs (e.g., ChatGPT, Claude and DeepSeek), allowing seamless switching without altering core logic.

On the tool side, the system maintains a dynamic set of modules as follows:

$$\mathcal{T} = \{\mathcal{T}_{\text{ML}}, \mathcal{T}_{\text{OCR}}, \mathcal{T}_{\text{unit}}, \cdots\},$$

each registered with standardized interfaces for plug-and-play integration. Researchers can customize pipelines by adding domain-specific tools, preprocessing functions or validation tests.

This architecture enables flexible adaptation to diverse tasks, from general-purpose modeling to specialized workflows such as time series forecasting or biomedical analysis–making `VDSAgents` a customizable and portable foundation for automated data science systems.

3.4 | Agent Function Interface and Mapping

Each agent in the `VDSAgents` is equipped with a set of structured functions $\mathcal{F}_{\mathcal{A}_i} = \{f_1, f_2, \ldots, f_m\}$, enabling it to perform domain-specific reasoning, code generation and intermediate decision-making. These functions can be called using natural language prompts and operate within a unified context composed of system messages, task instructions and memory states.

The behaviour of any agent $\mathcal{A}_i$ can be formalized as a functional mapping:

$$\mathcal{A}_i: \quad (\mathcal{S}_{\text{context}}, \mathcal{F}_{\mathcal{A}_i}) \to \mathcal{R}_{\text{task}},$$

where

- $\mathcal{S}_{\text{context}}$ represents the accumulated upstream outputs and task state;
- $\mathcal{F}_{\mathcal{A}_i}$ is the agent's internal callable function set;
- $\mathcal{R}_{\text{task}}$ is the resulting code, outputs, or structured reasoning reports.

Each function is designed to be modular, interpretable and robust in perturbation. Supporting Information Appendix B.3 provides detailed function listings for key agents.

4 | Experiments and Evaluation

4.1 | Experimental Setup

4.1.1 | Dataset

To evaluate the stability, robustness and predictive performance of the proposed `VDSAgents`, we carry out experiments on nine representative datasets. These datasets range from clean, preprocessed data to raw data, allowing us to assess generalizability.

Let $\mathcal{D} = \{D_1, D_2, \ldots, D_9\}$ denote the dataset collection, categorized as follows:

- **Clean datasets** ($\mathcal{D}_{\text{clean}}$): This group includes `bank_churn`, `titanic` and `obesity_risks`, sourced from https://www.kaggle.com. These datasets are already partially processed, with low missingness and consistent logical structures.
- **Raw datasets** ($\mathcal{D}_{\text{raw}}$): Including `adult`, `In-Vehicle_Coupon_Recommendation`, `parkinsons` and `Seoul_Bike_Sharing_Demand`, these are drawn from the UCI machine learning repository. They feature minimal preprocessing and present more realistic challenges, such as missing data and noisy attributes.
- **High-dimensional complex datasets** ($\mathcal{D}_{\text{complex}}$): Consisting of `ames_houses` and `online_shopping`, both from https://vdsbook.com/, these datasets are used in real-world educational or applied settings and involve intricate combinations of continuous and categorical features.

These datasets enable a comprehensive evaluation of `VDSAgents`' capabilities across various scenarios, particularly in tasks such as identification of response variables, selection of the feature pipeline, model comparison and evaluation of the stability based on perturbations. The complete dataset details are provided in the Supporting Information Appendix D.

4.1.2 | Evaluation Metrics

To systematically evaluate the performance of VDSAgents across diverse tasks and perturbed scenarios, we adopt the evaluation protocol introduced by Hong et al. (2024), incorporating additional considerations regarding task performance and completion quality. Three core metrics are defined below:

- **Valid Submission (VS)**: Measures the proportion of attempts in which the system successfully generates a syntactically correct, executable and evaluable pipeline:

$$\mathrm{VS} = \frac{T_s}{T},$$

where T_s is the number of successful attempts and T is the total number of attempts. For each experiment, we repeatedly run the system until five valid outputs are obtained ($T_s = 5$). As model responses may fail or be invalid, the value of T varies across runs. Thus, $VS = \frac{5}{T}$, enabling indirect inference of total attempts from reported VS values.

- **Average Normalized Performance Score (ANPS)**: We first compute the Normalized Performance Score (NPS) for each valid run and then report the mean and standard deviation across all N valid runs. This "run-level-first" design enhances metric robustness and enables meaningful statistical reporting.

$$\mathrm{NPS} = \begin{cases} \frac{1}{4}(\mathrm{Accuracy} + \mathrm{F1} + \mathrm{Precision} \\ \qquad + \mathrm{Recall}), & \text{for classification tasks,} \\ \frac{1}{3}\left(\frac{1}{1+\mathrm{RMSE}} + \frac{1}{1+\mathrm{MAE}} + R^2\right), & \text{for regression tasks.} \end{cases}$$

$$\mathrm{ANPS} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{NPS}^{(i)}, \quad \mathrm{SD}_{\mathrm{ANPS}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\mathrm{NPS}^{(i)} - \mathrm{ANPS}\right)^2}.$$

This 'run-level-first" approach enhances metric robustness and enables meaningful statistical reporting. For classification tasks, ANPS values are always between 0 and 1. For regression tasks, ANPS may become negative if $R^2_{\mathrm{avg}} < 0$, which can occur in high-noise settings where model performance falls below baseline.

- **Comprehensive Score (CS)**: Combines execution robustness and modeling performance into a unified metric as follows:

$$\mathrm{CS} = 0.5 \times \mathrm{VS} + 0.5 \times \mathrm{ANPS}.$$

Equal weights are assigned to validity and quality, making CS suitable for comparing systems under heterogeneous data science tasks.

4.1.3 | Baselines and Model Configurations

To benchmark the performance of VDSAgents, we compare it with two representative multi-stage automated data science frameworks: AutoKaggle (Li et al. 2024) and DataInterpreter (Sun et al. 2024). For each system and dataset, experiments are repeated multiple times until five successful runs with valid outputs are obtained, and the final reported performance is averaged over these five runs.

We test both systems under two widely used LLM backends:

- **GPT-4o**: A state-of-the-art OpenAI model with strong reasoning and code generation capabilities.
- **DeepSeek-V3**: A competitive open-source model representing leading domestic performance in structured tasks.

For all EDA-related visual analysis, we employ **Qwen-VL-7B** as the unified image-to-text module $\mathcal{T}_{\mathrm{OCR}}$ in both systems. We fix the maximum self-repair steps at $N_{\max} = 3$ and set the number of perturbations to $k = 50$.

4.2 | Results and Analysis

This section presents a comprehensive evaluation of VDSAgents compared to AutoKaggle and DataInterpreter on nine benchmark tasks, comprising six classification datasets and three regression datasets. The comparison focuses on three core dimensions that we have defined: VS, ANPS and CS. Supporting Information Table 8 summarizes the performance in nine system configurations, each using one of two LLM back-ends: DeepSeek-V3 and GPT-4o.

4.2.1 | Execution Stability

VS measures the proportion of trials where the system successfully produces a valid, executable and predictive output. As shown in Supporting Information Table 8, VDSAgents consistently achieves higher execution stability compared to AutoKaggle and DataInterpreter. Specifically, with DeepSeek-V3 and GPT-4o, VDSAgents attains average VS scores of 0.894 and 0.950, respectively, clearly outperforming AutoKaggle (0.577 and 0.534) and DataInterpreter (0.676 and 0.672).

Notably, VDSAgents with GPT-4o achieves 100% success in eight of nine tasks, demonstrating strong robustness and compatibility with advanced LLMs. In contrast, both baseline methods exhibit lower and more variable performance, especially in challenging regression scenarios such as parkinsons and online_shopping datasets.

Overall, VDSAgents offers a significantly more reliable execution framework for diverse tasks and conditions.

4.2.2 | Predictive Effectiveness

ANPS reflects average predictive quality conditional on successful execution, combining classification accuracy and regression effectiveness. As shown in Supporting Information Table 8, VDSAgentsconsistently achieves higher ANPS scores than both AutoKaggle and DataInterpreter. With GPT-4o,

VDSAgents achieves an average ANPS of 0.692, clearly outperforming AutoKaggle (0.497) and DataInterpreter (0.569). Similarly, with DeepSeek-V3, VDSAgents obtains 0.667, surpassing AutoKaggle (0.599) and DataInterpreter (0.588).

In classification tasks, the three systems show relatively similar performance, with the average ANPS generally ranging between 0.7 and 0.9. However, VDSAgents maintains slightly more stable and higher performance overall, particularly evident in datasets with noise or high dimensionality (e.g., online shopping).

In regression tasks, performance divergence is more significant. VDSAgents with GPT-4o excels substantially (Parkinson's ANPS=0.947, Seoul Bike ANPS=0.237), indicating its superior capability to model continuous numerical outputs. In contrast, both AutoKaggle and DataInterpreter frequently deliver poor results, with AutoKaggle-GPT4o even yielding negative scores (e.g., Ames Houses), revealing clear limitations in numerical modeling capability and pipeline robustness.

These trends underscore the advantage of VDSAgents, particularly in regression contexts involving continuous variables, noise, or high dimensionality, where baseline methods struggle significantly.

Further details including NPS variability and ANPS visualizations with error bars are provided in the Supporting Information Appendix E.

4.2.3 | Overall Capability

To jointly evaluate the stability and robustness of the execution and predictive effectiveness, we define CS as the average of VS and ANPS. As shown in the Supporting Information Table 8 and Figure 3, VDSAgents consistently achieves the highest CS scores compared to both AutoKaggle and DataInterpreter. Specifically, VDSAgents-GPT4o attains an average CS of 0.821, clearly surpassing AutoKaggle-GPT4o (0.515) and DataInterpreter-GPT4o (0.621). Similar advantages persist with DeepSeek-V3.

In general, these results underscore the clear superiority of VDSAgents, offering robust execution combined with effective predictions in diverse tasks and challenging conditions.

4.2.4 | Ablation Study: Impact of PCS-Agent

To further understand the contribution of key components within VDSAgents, we conducted an ablation study focusing on the role of the PCS-Agent–responsible for verifying predictability, computability and stability properties during pipeline generation.

We selected two representative datasets: Online Shopping (classification) and Ames Housing (regression). We compared performance with and without the PCS-Agent module under the same evaluation protocol.

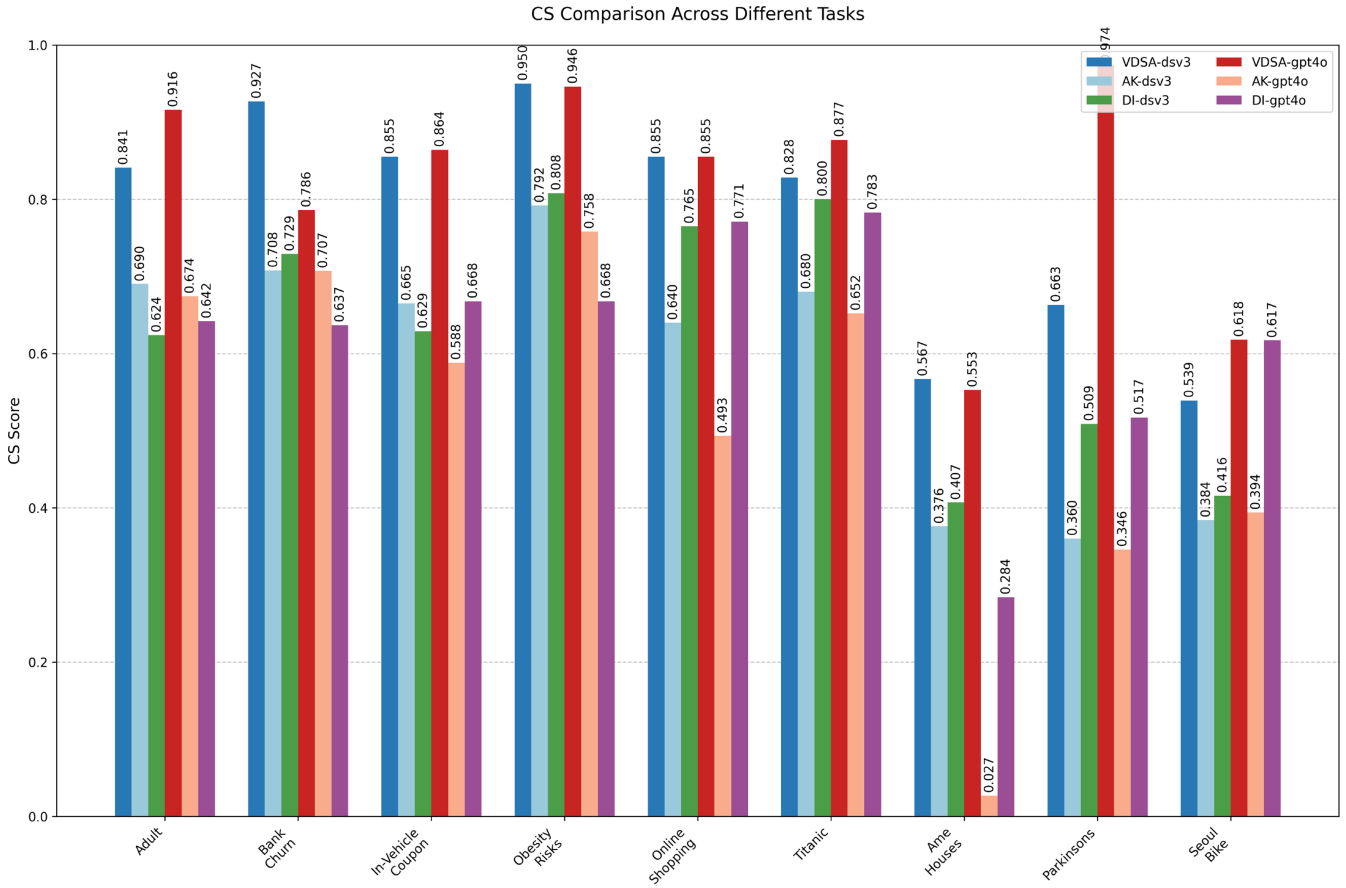


FIGURE 3 | Comparison of comprehensive score (CS) across four system variants on nine tasks.

As shown in the Supporting Information Table 9, removing the PCS-Agent results in substantial performance degradation, especially on the classification task (VS ↓ 20%, ANPS ↓ 49%, CS ↓ 31%). This demonstrates the importance of PCS auditing in maintaining robustness and predictive effectiveness across diverse tasks.

These results reinforce the critical role of PCS validation in improving both system reliability and predictive robustness.

For additional ablation results on key hyperparameters k and N_{max}, please refer to Supporting Information Appendix F.

5 | Discussion

The superior performance of VDSAgents over AutoKaggle and DataInterpreter is attributable not only to more powerful LLM back-ends but also to its principled design grounded in PCS principles. VDSAgents emphasizes three key elements of PCS: the ability to generate models that generalize (predictability), execute reliably (computability) and remain robust under perturbations (stability). These principles are encoded in the logic of the VDSAgents at both the system and function levels. Agents are not passive responders to prompts, but active planners that construct problem-solving trajectories aligned with the structure of the data and the goals of the analysis.

A concrete manifestation of this structure is observed in the way the system performs data cleaning. For example, in missing value imputation, AutoKaggle typically applies global methods (e.g., filling with column-wise mean), overlooking latent data hierarchies. This is a common pitfall in datasets where group structure matters, such as time series per stock or country-level survey data, where such methods can distort distributions and mislead downstream models.

In contrast, VDSAgents, under PCS-Guided, decomposes the cleaning process into reasoning steps: It first identifies the semantic roles of variables (e.g., StockCode, Country), then proposes targeted imputation strategies (e.g., per-stock mean, per-region median) that align with domain structure. These strategies are not hallucinated, but supported by executable modular functions from the mltools library.

Furthermore, the PCS-Agent operationalizes stability by generating perturbed variants of the data based on alternative, yet plausible, structural assumptions, such as imputing by continent instead of country. This enables the system to systematically assess the sensitivity of modeling outcomes, a core idea of the PCS principles that is often neglected in black-box pipelines.

By aligning each phase of the workflow, cleaning, modeling, evaluation, with a PCS-Guided structure, VDSAgents avoids the brittleness of purely prompt-driven systems. It produces results that are not only accurate but also reproducible, interpretable and robust to variation. In short, VDSAgents is not just "LLM powered" but "PCS-Guided", and this distinction is central to its observed advantages in diverse tasks and data types.

6 | Conclusion and Future Work

This paper introduces VDSAgents, a modular, PCS-Guided, multi-agent automated data science framework. Guided by the core principles of predictability, computability and stability, the system decomposes tasks into structured agent workflows, integrates reusable tools and enables scientifically grounded modeling in real-world data scenarios.

Empirical results in nine datasets demonstrate the effectiveness of our design. VDSAgents achieves superior execution stability, robust predictive performance and leading overall capability, outperforming the baseline AutoKaggle and DataInterpreter under both the GPT-4o and DeepSeek-V3 backends. Its performance is especially notable on complex and noisy datasets, where its structured inference paths and stability-driven evaluation yield consistent gains.

Several promising directions remain open for extending the capabilities of VDSAgents:

- **Fine-grained stability modeling:** Beyond basic data cleaning and feature selection, future work could explore stability-aware designs for more advanced modeling paths such as causal inference (Wang et al. 2025) and multitask learning (Agarwal et al. 2025).
- **Human-in-the-loop feedback:** Integrating expert feedback at key decision points could enable adaptive refinement of strategies and improve performance in domain-specific tasks.
- **Cross-domain generalization:** Applying the PCS-Guided architecture to critical domains such as healthcare, finance or policy analysis will help evaluate its transferability and practical value under higher reliability demands.

By combining theoretical guidance with system-level engineering, VDSAgents offers a trustworthy foundation for LLM-driven data science. We envision its broader applications in intelligent research, automated analysis and education, helping bridge the gap between automation and scientific reasoning in data-driven practice.

Author Contributions

Yunxuan Jiang and Silan Hu contributed equally to this work. Yunxuan Jiang led the overall manuscript writing. Silan Hu and Yunxuan Jiang jointly developed the experimental code, discussed and refined the framework details, and improved the open–source GitHub repository. Xiangyu Chang served as the corresponding author and supervising advisor, providing guidance to the co–first authors on experiment design, implementation, and manuscript preparation. Xiaoning Wang and Yuanyuan Zhang provided domain expertise, helping align our framework with leading industry projects and offering relevant guidance. All authors reviewed and approved the final manuscript.

Data Availability Statement

The data that support the findings of this study are available in Adult Income Prediction at https://archive.ics.uci.edu/dataset/560/seoul+;. These data were derived from the following resources available in the public domain: Bank Customer Churn (https://www.kaggle.com/competitions/playground-series-s4e1/code), In-Vehicle Coupon

Recommendation (https://archive.ics.uci.edu/dataset/603/in+vehicle+coupon+recommendation), Obesity Risk Prediction (https://www.kaggle.com/competitions/playground-series-s4e2), Online Shopping Intentions (https://vdsbook.com/), Titanic Survival Prediction (https://www.kaggle.com/competitions/titanic), Ames Housing Prices (https://vdsbook.com/), Parkinson's Telemonitoring (https://archive.ics.uci.edu/dataset/189/parkinsons+), and Seoul Bike Sharing (Demand, https://archive.ics.uci.edu/dataset/560/seoul+).

Endnotes

[1] VDSAgents projection is publicly available at https://github.com/fengzer/VDSAgents.

References

Achiam, J., S. Adler, S. Agarwal, et al. 2023. "GPT-4 technical report." *arXiv preprint arXiv:2303.08774*.

Agarwal, A., M. Xiao, R. Barter, O. Ronen, B. Fan, and B. Yu. 2025. "PCS-UQ: Uncertainty Quantification via the Predictability-Computability-Stability Framework." *arXiv preprint arXiv:2505.08784*.

Cai, C., L. Bai, S. Zha, and E. Chen. 2023. "LLMs as Tool Makers: Teaching LLMs to Create and Use Tools." *arXiv preprint arXiv:2312.06644*.

Cao, L. 2017. "Data Science: A Comprehensive Overview." *ACM Computing Surveys (CSUR)* 50, no. 3: 1–42. https://doi.org/10.1145/3076253.

DeepSeek-AI. 2024. "DeepSeek LLM: Scaling Open-Source Language Models with Longtermism." *arXiv preprint arXiv:2401.02954*.

Gao, L., A. Madaan, S. Zhou, et al. 2023. "PAL: Program-Aided Language Models." In *Proceedings of the 40th International Conference on Machine Learning*, vol. 202, 10764–10799.

Hong, S., Y. Lin, B., Liu, et al. 2024. "Data Interpreter: An LLM Agent For Data Science." *arXiv preprint arXiv:2402.18679*.

Jiang, Z., D. Schmidt, D. Srikanth, et al. 2025. "AIDE: AI-Driven Exploration in the Space of Code." *arXiv preprint arXiv:2502.13138*.

Li, Z., Q. Zang, D. Ma, et al. 2024. "AutoKaggle: A Multi-Agent Framework for Autonomous Data Science Competitions." *arXiv preprint arXiv:2410.20424*.

Madaan, A., N. Tandon, P. Gupta, et al. 2023. "Self-Refine: Iterative Refinement With Self-Feedback." In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, vol. 36, 46534–46594.

Provost, F., and T. Fawcett. 2013. "Data Science and Its Relationship to big Data and Data-Driven Decision Making." *Big Data* 1, no. 1: 51–59. https://doi.org/10.1089/big.2013.1508.

Qian, Y., S. Chen, Y. Jiang, et al. 2023. "Creator: A Unified Agent Framework for Tool Creation and Usage." *arXiv preprint arXiv:2310.01753*.

Schick, T., J. Dwivedi-Yu, L. von Werra Sun, et al. 2023. "Toolformer: Language Models Can Teach Themselves to Use Tools." *arXiv preprint arXiv:2302.04761*.

Sun, M., R. Han, B. Jiang, et al. 2024. "LAMBDA: A Large Model Based Data Agent." *arXiv preprint arXiv:2407.17535*.

Technology, T. 2024. "Integrating Unit Testing into Data Science Lifecycle." Medium. Published December 12, 2024. Accessed July 25, 2025. https://medium.com/@turkishtechnology/integrating-_unit-_testing-_into-_data-_science-_lifecycle-_6de01a33a4c0.

Tu, X., J. Zou, W. J. Su, and L. Zhang. 2024. "What Should Data Science Education Do With Large Language Models?" *Harvard Data Science Review* 6, no. 1. https://doi.org/10.1162/99608f92.bff007ab.

Wang, B., X. Lin, P. Liu, Z. S. Liu, and Y. Liu. Code-as-Policies: Self-Debugging LLMs with Unit Tests. *arXiv preprint arXiv:2305.13242*, 2023.

Wang, Q., T. M. Tang, M. Youlton, et al. 2025. "Epistasis Regulates Genetic Control of Cardiac Hypertrophy. *Nature*." *Cardiovascular Research* 1–21.

Yao. S., D. Yu, J. Zhao, B. Yu, and K. Narasimhan. 2023. "Tree of Thoughts: Deliberate Problem Solving with Large Language Models." *arXiv preprint arXiv:2305.10601*.

Yao, S., J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. 2022. "ReAct: Synergizing Reasoning and Acting in Language Models." *arXiv preprint arXiv:2210.03629*.

Yu, B. 2013. "Stability." *Bernoulli* 19, no. 4: 1484–1500. https://doi.org/10.3150/13-_BEJSP14.

Yu, B. and R. L. Barter. 2024. "Veridical Data Science: The Practice of Responsible Data Analysis and Decision Making." Accessed May 15, 2025. https://vdsbook.com/.

Yu, B., and K. Kumbier. 2020. "Veridical data science." *Proceedings of the National Academy of Sciences* 117, no. 8: 3920–3929. https://doi.org/10.1073/pnas.1901326117.

Zhou, Y., Y. Zhang, Y. Xu, Z. Yang, Z. Lin, L. Pan, and Z. Liu. "AutoAgents: Self-improving LLM Agents via Uncertainty and Self-Reflection." *arXiv preprint arXiv:2309.00661*, (2023).

Zhou, Z., Y. Tang, Z. Yuan, C. Xu, Z. Liu, J. Tang, and J. Zhou. 2023. "White-box Testing for LLMs via Execution Verification." *arXiv preprint arXiv:2307.03868*.

Supporting Information

Additional supporting information can be found online in the Supporting Information section. Supplementary_Material.pdf appendix.tex