JUPITER: Enhancing LLM Data Analysis Capabilities via Notebook and Inference-Time Value-Guided Search

Shuocheng Li[1*], Yihao Liu[1*], Silin Du[2*], Wenxuan Zeng[1*], Zhe Xu[1*], Mengyu Zhou[3] Yeye He[3], Haoyu Dong[3], Shi Han[3], Dongmei Zhang[3]

[1]Peking University
[2]Stanford University
[3]Microsoft
mengyu.chou@gmail.com

Abstract

Large language models (LLMs) have shown great promise in automating data science workflows. However, existing models still struggle with multi-step reasoning and tool use, limiting their effectiveness on complex data analysis tasks. To address this limitation, we propose a scalable pipeline that extracts high-quality, tool-based data analysis tasks and their executable multi-step solutions from real-world Jupyter notebooks and associated data files. Using this pipeline, we introduce NbQA, a large-scale dataset of standardized task–solution pairs that reflect authentic tool-use patterns in practical data science scenarios. To further enhance the multi-step reasoning capabilities, we present JUPITER, a framework that formulates data analysis as a search problem and applies Monte Carlo Tree Search (MCTS) to generate diverse solution trajectories for value model learning. During inference, JUPITER combines the value model and node visit counts to efficiently collect executable multi-step plans with minimal search steps. Experimental results show that Qwen2.5-7B and 14B-Instruct models on NbQA solve 77.82% and 86.38% of tasks on InfiAgent-DABench, respectively—matching or surpassing GPT-4o and advanced agent frameworks. Further evaluations demonstrate improved generalization and stronger tool-use reasoning across diverse multi-step reasoning tasks.

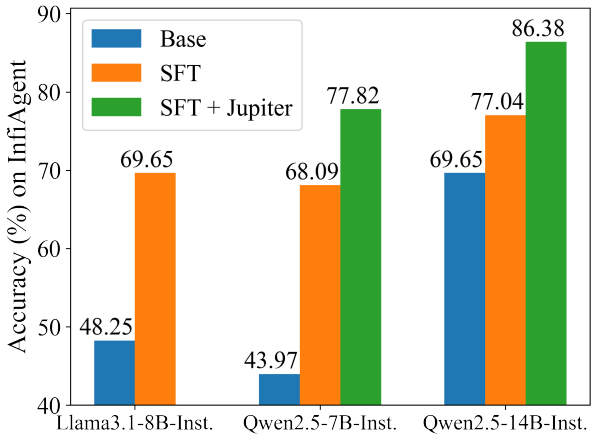

Figure 1: Accuracy by Questions (%) across different models on InfiAgent-DABench under three settings: base model, SFT on **NbQA**, and SFT on **NbQA** + JUPITER search, demonstrating the base models suffer from accurate multi-step data analysis.

Code — https://github.com/microsoft/Jupiter
Datasets — https://github.com/microsoft/Jupiter
Extended version — https://arxiv.org/abs/2509.09245

Introduction

A complete data analysis workflow (Donoho 2017) often involves multi-step derivations and decisions where intermediate steps are highly interdependent. This process demands that practitioners continuously explore and iteratively optimize their approaches, guided by feedback from their decisions. Furthermore, addressing specific scenarios in data analysis frequently requires domain knowledge and specialized tools. Therefore, designing a correct and practical data analysis workflow is a challenging and complex task, posing significant obstacles to achieving fully automated solutions (De Bie et al. 2022).

Recent advancements in large language models (LLMs) and LLM-based agents (Cheng, Li, and Bing 2023; Qiao et al. 2024; Hollmann, Müller, and Hutter 2023; Zhang et al. 2024) have demonstrated significant potential in automating certain data science tasks, such as feature engineering (Hollmann, Müller, and Hutter 2023), data visualization (Yang et al. 2024), and model selection (Shen et al. 2023).However, these methods typically focus on only a single stage of the data analysis process, without considering the interdependencies across real-world data science tasks. Some agent-based systems (e.g., Data Interpreter (Hong et al. 2024), AutoGen (Wu et al. 2023), Taskweaver (Qiao et al. 2024)) have attempted to provide more comprehensive solutions, but these approaches often rely on proprietary commercial models, complex task decomposition, and intricate system designs. Even the most advanced models—open or propri-

*Work done during internship at Microsoft.

etary—still face significant limitations in data understanding and multi-step reasoning with tools (Hu et al. 2024).

In this work, we aim to improve models' multi-step data analysis capabilities by combining large-scale notebook-derived solutions with inference-time value-guided search. We first construct **NbQA**, a dataset of 38k high-quality and diverse task–solution pairs extracted through a fine-grained processing pipeline that identifies analysis tasks and their corresponding multi-step code-based solutions from real Jupyter notebooks, with 6.8k samples providing full executable environments. Fine-tuning 7B and 14B models on NbQA yields substantial gains on InfiAgent-DABench, including over 21% absolute improvement for 7B models.

To further strengthen multi-step reasoning, we introduce JUPITER, which formulates notebook-based data analysis as a state-level search problem, where each node corresponds to a notebook execution state consisting of the generated code cell and its output. Using tasks with executable environments, we employ MCTS to collect diverse trajectories and train a value model to guide inference-time search. During inference, we disable the PUCT exploration term and rely on value estimates and visit counts to efficiently identify promising solution paths. With only 40 iterations, our value-guided search enables fine-tuned 7B and 14B models to reach 77.82% and 86.38% accuracy, respectively—matching or surpassing GPT-4o and strong agent frameworks.

We further demonstrate generalization on DSBench—a data modeling benchmark benchmark—and AIME, showing strong transferability across diverse tasks and enhanced numerical/tool-use reasoning in out-of-domain settings.

In summary, the contributions of our work are as follows:

- We propose an automated pipeline for extracting high-quality data analysis tasks and their multi-step solutions from large-scale Jupyter notebooks and associated data files. Using this pipeline, we construct **NbQA**, a large-scale, high-quality dataset of standardized task–solution pairs. NbQA covers diverse, practical data analysis scenarios, supporting both supervised fine-tuning and value model training.
- We propose JUPITER, a general framework that formulates notebook-based data analysis as a sequential decision-making search problem. JUPITER leverages a value model trained from interaction trajectories to guide inference-time search, efficiently discovering diverse candidate solutions under limited interaction steps.
- We conduct comprehensive experiments across data analysis and out-of-domain benchmarks. Our approach significantly improves multi-step reasoning, tool-use, and generalization abilities of 7B and 14B models, matching or surpassing strong commercial models such as GPT-4o on InfiAgent-DABench, and demonstrating strong transferability on DSBench and AIME.

Construction of the NbQA Dataset

In this section, we describe the workflow used to construct the NbQA dataset (see Figure 2). Details of the dataset construction can be found in the extended version.

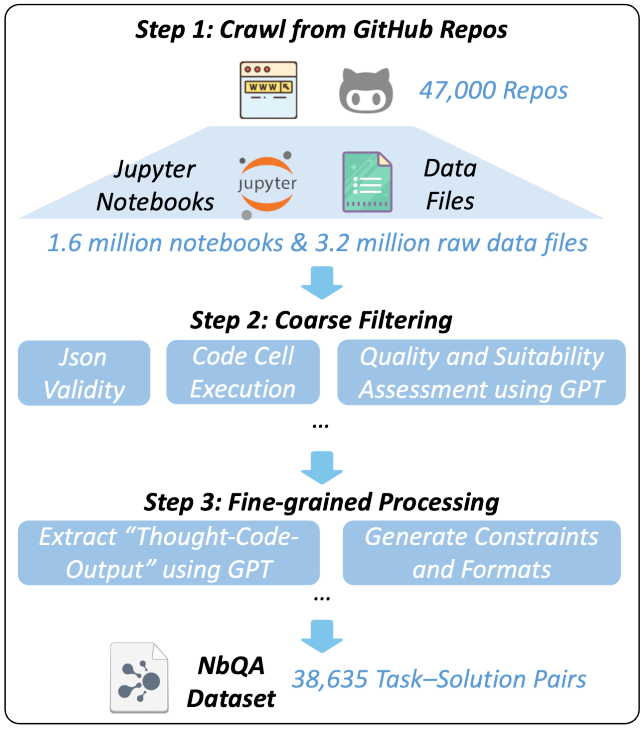

Figure 2: Construction of the NbQA dataset.

Crawl Notebooks on GitHub

We first use the GitHub API to collect Jupyter Notebooks with the `.ipynb` suffix from open-source repositories, identify data loading functions via regular expressions, and extract the required data files. We then search for and download these files from each repository's working directory. In total, we collect 1.6 million notebooks and 3.2 million raw data files from about 47,000 repositories. Notebooks with missing data dependencies are still retained for extracting task-solution pairs for supervised fine-tuning.

Coarse Filtering

To ensure data quality and reliability, we first validated all crawled Jupyter notebooks by checking their underlying JSON structure. We excluded notebooks that were not executed in code-cell order, contained unexecuted cells, or raised unhandled execution errors. To prevent data contamination and test set leakage, we performed keyword matching on filenames and notebook content to remove any notebooks referencing common teaching or competition datasets, such as Iris (Manimala 2024), Titanic (Baghizadeh 2024), Breast Cancer (Namdari 2024), Boston Housing (Schirmer 2024), and Wine Quality (Learning 2024). We also removed notebooks involving overly simple tasks, such as those processing data files with fewer than 20 rows or containing fewer than 40 lines of code.

As an additional quality control step, we prompted GPT-4o mini to assign a score from 1 to 5 for each notebook, based on factors such as code correctness and structure, data relevance and complexity, and the appropriate use of stan-

dard Python libraries. Only notebooks with a score of 3 or higher were retained.

To further ensure dataset quality and diversity, we applied additional filtering and deduplication strategies. We limited each GitHub repository to at most one retained notebook if its data files were incomplete, and at most two if complete. We also grouped notebooks by the hash values of their data dependency files and retained only those groups where all notebooks originated from the same repository, keeping at most two notebooks per group. Finally, to focus on classical data analysis tasks, we filtered out all notebooks involving neural networks, pre-trained models, or GPU-based computation.

Fine-grained Processing

To ensure that the dataset is thematically focused and structurally consistent, we begin by using GPT-4o mini to identify the machine learning models present in each notebook. For notebooks involving machine learning, we retain only those that utilize one or more of 21 commonly used classical models (e.g., Logistic Regression, Support Vector Machine, Linear Regression), which collectively cover major machine learning tasks.

We then define eight categories of data analysis tasks: Summary Statistics, Distribution Analysis, Correlation Analysis, Outlier Detection, Comprehensive Data Preprocessing, Feature Engineering, Machine Learning, and Visualization. Guided by examples, we instruct GPT-4o to extract 1 to 3 representative subtasks from each notebook, along with their corresponding answers and task types. To ensure answer verifiability, we require that, except for visualization tasks, all numeric answers must be directly extractable from the notebook's code outputs. The prompt also enforces strict constraints: extracted tasks must be well-defined, free of contextual assumptions, and must not merely restate code execution results.

To further address potential issues such as vague task descriptions or inconsistent answer formats, we instruct the model to supplement each task with explicit solution constraints and standardized output formats, and to present answers in the form of `@answer_name[answer]` labels to facilitate automatic evaluation. We then prompt the model to generate multi-step solutions for each task. Unlike previous synthetic data approaches that rely on LLM-generated solutions, our method does not generate answers from scratch but instead extracts relevant code and outputs directly from the original notebook. The model is strictly prohibited from fabricating outputs or skipping steps, ensuring that each solution —comprising multiple `thought-code-output` blocks and a final `thought-label` block—faithfully reflects the actual analysis workflow.

Finally, we apply automated review using GPT-4o mini to filter out tasks with mismatched answers or vague constraints. After all filtering steps, the final NbQA dataset contains 38,635 task–solution pairs, of which 6,845 are associated with complete and fully interactive data dependency files and exhibit low randomness suitable for automatic evaluation.

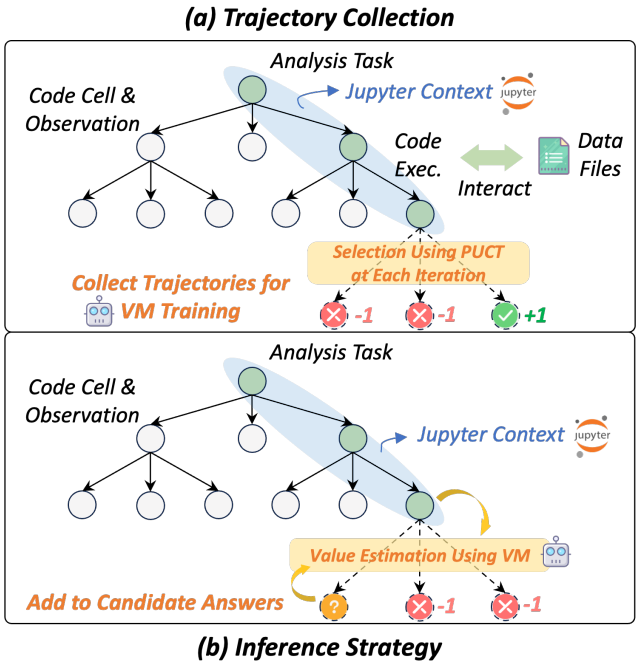

Figure 3: Overview of JUPITER, including (a) trajectory collection and (b) inference strategy. The collected trajectories are used for training the value model (VM), and VM is used for value estimation during inference.

JUPITER Framework

In this section, we demonstrate how the JUPITER framework transforms solving data analysis problems using notebooks into a search problem. The full procedure is illustrated in Figure 3 and detailed in Algorithm 1. Then we briefly describe the structure and training process of the value model.

Problem Formulation

We formalize multi-step data analysis as a sequential decision-making search problem. The process is modeled as a tree, where the root node represents the initial notebook state s_0—including the data analysis problem to be solved—and each edge corresponds to the generation and execution of a code cell. Each node in the tree records the current thought, code, and the resulting output, while the path from the root to any node forms the complete notebook context at that step.

At each search step, the agent selects a node to expand using the PUCT algorithm and samples K candidate thought-action pairs from the language model. Each candidate is executed in the current notebook context to create a new child node. Terminal nodes represent answers or error states, while intermediate nodes can be further expanded. Rewards are backpropagated along the tree to affect the search process. Specifically, for each node s', the PUCT score is calculated as follows:

$$\mathrm{PUCT}(s') = Q(s') + c_{\mathrm{puct}} \cdot P(s') \cdot \frac{\sqrt{N_{\mathrm{parent}}(s')}}{1 + N(s')} \quad (1)$$

Algorithm 1: JUPITER for Notebook-based Data Analysis

Require: Initial notebook state s_0, value model V_ϕ (optional), search budget N, number of expansions per step K

Initialize search tree with root node s_0
for iteration $= 1$ **to** N **do**
  if Trajectory Collection Phrase **then**
    Select node to expand using PUCT ($c_{puct} > 0$)
  else
    Select node to expand using PUCT ($c_{puct} = 0$)
  end if
  Sample K candidate thought-action pairs from LLM
  for each candidate **do**
    Execute code in notebook context, obtain output
    Expand notebook content to create a new child node
    if Trajectory Collection Phrase **then**
      if Child node is terminal **then**
        Assign reward ($+1$ if correct answer, -1 if error/invalid)
      else
        Estimate value using V_ϕ if available
      end if
    else
      Estimate value using V_ϕ for this child node
    end if
    Backpropagate reward or value along the path
  end for
end for
Return candidate answer nodes collected during search

where $Q(s')$ is the average value estimate of node s', $P(s')$ is the prior probability assigned by the language model, $N_{\text{parent}}(s')$ is the visit count of the parent node, $N(s')$ is the visit count of node s' itself, and c_{puct} is a hyperparameter controlling the tradeoff between exploitation term $Q(s')$ and exploration term $P(s') \cdot \frac{\sqrt{N_{\text{parent}}(s')}}{1+N(s')}$. The node with the highest PUCT score is selected for expansion at each step.

During the trajectory collection phase, terminal nodes are assigned positive or negative rewards for correctness. For each node, its $Q(s')$ is estimated by averaging the returns from the observed rewards of its descendant nodes during the search. The collected trajectories, along with these Q values, are then used to train a value model that predicts the expected return for each notebook state. In the inference phase, the exploration term in PUCT is removed (i.e., $c_{puct} = 0$), and node selection relies primarily on value estimates from the trained value model for both intermediate nodes and candidate answer nodes, focusing the search on promising branches and improving efficiency. This design is motivated by the fact that data analysis tasks have a vast, sparse search space, where most branches are invalid, while correct solutions are concentrated in a few high-quality branches. Retaining the exploration term causes unnecessary exploration and computation. In contrast, a well-trained value model effectively identifies promising nodes, enabling more focused and efficient search—a result also supported by our experiments.

The search proceeds until a maximum number of iterations or a stopping criterion is reached, and all candidate answer nodes encountered during the search are retained. For tasks with reference answers, the final answer can be determined by majority voting or by selecting the candidate with the highest value estimate; for open-ended tasks, other task-specific evaluation metrics such as validation set accuracy can be used.

Model	Accuracy by Questions /%
Mistral-7B-Instruct-v0.3	**+56.81** (2.33 → 59.14)
Llama-3.1-8B-Instruct	**+21.40** (48.25 → 69.65)
Qwen2.5-7B-Instruct	**+24.12** (43.97 → 68.09)
QWen2.5-14B-Instruct	**+7.39** (69.65 → 77.04)

Table 1: Accuracy by Questions on InfiAgent-DABench before and after finetuning on NbQA.

Value Model Training

To train the value model, we first extract both successful and failed reasoning trajectories collected from 6,845 samples in NbQA during the Trajectory Collection phase. For each node along a trajectory, we use its corresponding context as input and the Q-value obtained from multiple MCTS simulations as the supervision signal. The value model is built upon the base language model fine-tuned on NbQA, with an additional regression-based value head attached. The value head outputs a scalar value in the range $[-1, 1]$ by pooling the final hidden states, and is trained using mean squared error loss against the normalized MCTS Q-values. Once trained, the value model provides accurate value estimation to guide tree expansion during inference, leading to more efficient and effective search. Experimental settings are provided in the extended version.

Experiments

We first present the performance improvements brought by supervised fine-tuning (SFT) of several models on the InfiAgent-DABench benchmark. We then highlight the further gains achieved by value-guided search in JUPITER, comparing it against other popular inference strategies and agent frameworks. Next, we evaluate the generalization ability of our approach on DSBench and AIME, demonstrating both the transferability of the value model across diverse data analysis task formats and the enhancement in tool-use capability after fine-tuning. For more detailed descriptions of the experimental setup and the effects of different hyperparameter values, please refer to the extended version.

Finetuning Results

We randomly sampled 8,975 instances from the NbQA, ensuring that benchmark data was excluded from both the supervised fine-tuning and trajectory collection phases, to construct a multi-turn interaction fine-tuning dataset following the ReAct-style (Yao et al. 2023b) format. The input prompt

includes the task, constraints, format, and required data files. The model is instructed to generate a *thought* and an *action* for each round of iteration based on the current context. The extracted python code block in *action* is executed by the Jupyter kernel of SandboxFusion (Bytedance-Seed-Foundation-Code-Team et al. 2025), and the resulting output is incorporated back into the history. This process continues until the model determines that sufficient information has been collected to output the formatted answer. We set the temperature to 0.2 and maximum iteration to 25, as required by (Hu et al. 2024).

As shown in Table 1, After 3 epochs of LoRA finetuning (Hu et al. 2021), the performance of 7B models on InfiAgent-DABench improves by more than 21% in accuracy by questions. This demonstrates that our constructed dataset effectively enables the model to enhance data analysis knowledge and multi-turn reasoning abilities.

InfiAgent-DABench

We compare JUPITER with several state-of-the-art agent systems and inference strategies on the InfiAgent-DABench benchmark, including Majority Voting (Wang et al. 2023), ReAct (Yao et al. 2023b), AutoGen (Wu et al. 2023), Taskweaver (Qiao et al. 2024), and Data Interpreter (Hong et al. 2024). For search and Majority Voting, we adopt the same prompt and multi-turn interaction format as used during fine-tuning. The maximum search tree depth is set to 10, with a maximum of 40 iterations. At each iteration, three sequences are sampled to expand the selected node. Each search path allows up to three code execution errors. For intermediate nodes and candidate answer nodes encountered during search, if the value model is used, value estimates are obtained from the value model and backpropagated; otherwise, a value of 0 is assigned and backpropagated. Finally, all candidate answer nodes are collected, and the final answer is determined by taking the mode among them. For Majority Voting, the temperature is set to 0.7. For each question, a maximum of 25 iterations is allowed, and five candidate solutions are sampled. For JUPITER, the temperature is also set to 0.7. For agent frameworks using GPT-4o, GPT-4o mini, and Qwen2.5-72B-Instruct, results are reported by (You et al. 2025), with the maximum number of iterations set to 21. The temperature for ReAct is set to 0.2, and for all other agents, the temperature is set to 0.

As shown in Table 2, JUPITER achieves its highest accuracy on InfiAgent-DABench when both the value model is used and the exploration term is removed. In this setting, the fine-tuned Qwen2.5-14B-Instruct reaches an ABQ of 86.38%, surpassing the best result of Taskweaver with GPT-4o, while outperforming all other open-source and proprietary agent baselines even when these use larger or commercial models like GPT-4o. The 7B model with the same JUPITER configuration also shows obvious improvement.

The results further show that introducing the value model significantly boosts accuracy, and removing the exploration term leads to the best performance. When the exploration term is present, the model explores more but at the cost of lower accuracy, indicating that a strong value model alone is sufficient to efficiently guide the search. In contrast, other

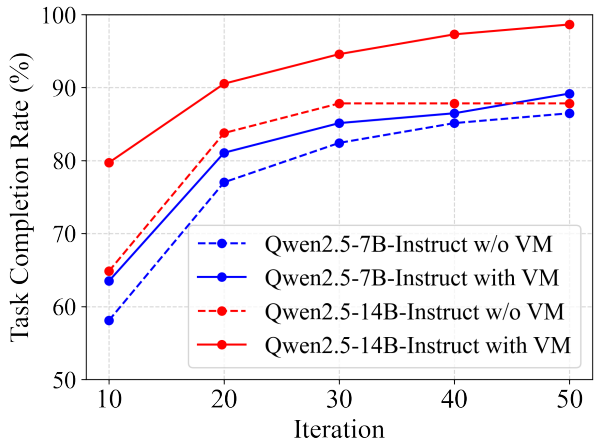

Figure 4: Task completion rates of QWen2.5-7B-Instruct and QWen2.5-14B-Instruct on the DSBench data modeling task as the number of iterations increases. "w/o VM" and "with VM" denote whether a trained value model is used. For comparison, when the maximum number of iterations is limited to 40, the task completion rates of QWen2.5-7B-Instruct and QWen2.5-14B-Instruct using ReAct are 63.51% and 66.22%, respectively.

frameworks—even with strong base models, fail to achieve this level of accuracy.

Overall, these findings suggest that value-guided search in JUPITER provides a general and effective means of enhancing multi-step data analysis for large language models, allowing open-source models to achieve performance competitive with the best commercial agent systems.

DSBench

We use the data modeling tasks from DSBench to evaluate the generalization ability of our trained value model across diverse formats of modeling tasks. All tasks focus exclusively on data modeling, without covering other types of data analysis. The data modeling tasks in DSBench are adapted from Kaggle competitions, where models are required to train machine learning models on provided training data, make predictions on the test set, write the predictions to a submission file, and compare the submission file to the ground-truth answer files to compute the task score. This process is quite different from the question-answering format used during the fine-tuning of our models. Therefore, we use vanilla QWen2.5-7B-Instruct and QWen2.5-14B-Instruct models, but still leverage our trained value model for value estimation during inference-time search. During the search, we set the maximum search tree depth to 25. For candidate answer files collected during the search process, as long as there exists at least one file that can pass the evaluation, the task is considered solved.

Additionally, we first prompt the model itself to remove redundant sections such as platform introductions, acknowledgements, and lists of companies and organizations from

Model	Framework	Acc. by Q. (%)
GPT-4o mini	ReAct	80.08
	Taskweaver	76.65
	AutoGen	70.04
	Data Interpreter	67.7
GPT-4o	ReAct	81.32
	AutoGen	73.54
	Taskweaver	85.99
	Data Interpreter	75.78
Qwen2.5-72B-Instruct	ReAct	75.88
	AutoGen	70.04
Qwen2.5-7B-Instruct *(SFT)*	ReAct	68.09
	Majority Voting (25 Iters, 5 Runs)	75.10
	JUPITER (40 Iters, w/o VM, w/o ExpTerm)	70.04
	JUPITER (40 Iters, w/o VM, with ExpTerm)	68.87
	JUPITER (40 Iters, withVM, with ExpTerm)	68.87
	JUPITER (40 Iters, withVM, w/o ExpTerm)	77.82
Qwen2.5-14B-Instruct *(SFT)*	ReAct	77.04
	Majority Voting (25 Iters, 5 Runs)	83.66
	JUPITER (40 Iters, w/o VM, w/o ExpTerm)	79.38
	JUPITER (40 Iters, w/o VM, with ExpTerm)	75.88
	JUPITER (40 Iters, withVM, with ExpTerm)	74.71
	JUPITER (40 Iters, withVM, w/o ExpTerm)	**86.38**

Table 2: Accuracy by questions on InfiAgent-DABench across various model settings. Models marked with *(SFT)* are supervised fine-tuned on NbQA. "w/o VM" and "with VM" indicate whether a trained value model is used. "w/o ExpTerm" and "with ExpTerm" indicate whether the exploration term in the PUCT selection algorithm is disabled ($c_{puct} = 0$) or enabled ($c_{puct} = 1.25$), respectively. "Iters" denotes the maximum number of search iterations allowed per question, and "Runs" denotes the number of independently sampled candidate solutions per question.

the original Kaggle competition task descriptions, retaining only the key information required for the task. Experiments show that this step significantly improves performance on data modeling tasks. With the help of search, small models can even surpass the task completion rates of previous frameworks that use GPT-4 or GPT-4o as the base model, such as Code Interpreter (OpenAI 2025) and AutoGen. This additional finding demonstrates that most of the challenge in this benchmark lies in the excessive and redundant context of the direct task descriptions, rather than in the inability of small models to solve the data modeling tasks themselves. Details of the simplification approach, experimental setup are provided in the extended version.

As shown in Figure 4, with an increasing number of iterations, the task completion rate of the vanilla model without value model assistance plateaus, whereas the vanilla model with value model-assisted search continues to improve. Finally, when the number of iterations reaches 50, QWen2.5-14B-Instruct with value model-assisted search achieves a task completion rate of 98.65%, while QWen2.5-7B-Instruct achieves 89.19%, both surpassing QWen2.5-14B-Instruct without value model assistance. These results demonstrate that even without SFT, our trained value model can effectively assist search in different data analysis tasks, showing transferability and generalization capability.

AIME

We further evaluate our models on the AIME 2025 benchmark, which consists entirely of math competition problems and lies outside the model's training distribution. Notably, JUPITER remains unadapted for math-specific tasks and continues to use multi-turn tool-use prompts designed primarily for data analysis.

As shown in Table 3, the vanilla Qwen2.5-7B-Instruct model demonstrates extremely limited tool-use ability: the accuracy for solving with Python code execution is 0.00%, consistent with results reported in prior work (Mai et al. 2025). After SFT on, the JUPITER search achieves a voting accuracy of 13.3% on AIME2025, which matches the result of CoT with multiple sampling before SFT. This suggests that SFT on NbQA meaningfully enhances the model's multi-step tool-use and numerical reasoning abilities even in a fully out-of-domain math context. At the same time, the proportion of questions for which at least one correct candidate is found rises to 26.7%, higher than CoT multi-sampling. Introducing a value model to guide the search further increases this metric to 33.3%. This indicates that, even in completely out-of-domain settings, the value model can generalize to help the model discover more correct solutions among the candidates, increasing answer diversity, even if these correct answers are not frequent enough to dominate voting due to the SFT model limitations.

Model	Strategy	Accuracy (%)
Vanilla	PoT	0.00
	CoT	6.67
	CoT + Multiple Sampling	23.3 (OR) / 13.3 (Vote)
SFT	ReAct	10.0
	JUPITER (w/o VM)	26.7 (OR) / 13.3 (Vote)
	JUPITER (with VM)	33.3 (OR) / 20.0 (Vote)

Table 3: Accuracy comparison on AIME2025 between vanilla and finetuned QWen-2.5-7B Instruct models. "CoT" refers to using step-by-step reasoning without code execution. "PoT" denotes solving the problem via Python code execution. "OR" means a question is considered correct if any of the sampled answers is correct; "VOTE" refers to majority voting. "w/o VM" and "with VM" indicate whether a trained value model is used. For CoT Multiple Sampling, 5 samples are generated for each question. For JUPITER, the maximum search depth is set to 8, and the maximum number of iterations is 40.

Related Work

LLM-Based Agents for Data Analysis

Large language models have shown strong potential in automating a range of data science tasks, with encouraging progress reported in areas such as feature engineering (Hollmann, Müller, and Hutter 2023), data visualization (Yang et al. 2024), and model selection (Shen et al. 2023). Nonetheless, these efforts often target isolated stages of the data analysis workflow, overlooking the interconnected nature of data science processes and limiting their ability to provide comprehensive, end-to-end support. Several frameworks, including Taskweaver (Qiao et al. 2024), AutoGen (Wu et al. 2023), and Data Interpreter (Hong et al. 2024), have aimed to offer more integrated solutions. However, most of these frameworks rely on complex deployments and are typically based on closed-source commercial large language models and intricate task decomposition and system design. In practice, the data analysis capabilities of current language models are often limited or underestimated, particularly for complex, multi-step scenarios. To overcome these limitations, we introduce an automated pipeline and the NbQA dataset to extract and standardize high-quality, executable multi-step data analysis tasks from Jupyter notebooks. We further present JUPITER, a value-guided search framework designed to enhance the multi-step data analysis capabilities of large language models.

Value-Guided Tree Search

Value-guided tree search, especially in the form of Monte Carlo Tree Search (MCTS), is a well-established method for sequential decision-making and has been widely used in areas such as game playing (Silver et al. 2016), program synthesis (Brandfonbrener et al. 2024), and more recently, complex reasoning with large language models (Yao et al. 2023a; Świechowski et al. 2022). Recent advances like AlphaMath (Chen et al. 2024) integrate value models into the search process to improve reasoning and solution quality. These approaches directly inspired us to apply MCTS to notebook-style, multi-step data analysis workflows. Specifically, we use MCTS-generated search trajectories to train value models, which are then used at inference time to efficiently guide the search for candidate solutions. This integration enables models to achieve robust multi-step reasoning and tool-use performance on real-world data analysis tasks.

Conclusion

In this work, we construct both a dataset and a framework to enhance the data analysis capabilities of LLMs. We introduce NbQA, a large-scale dataset of real-world data analysis tasks with multi-step solutions, extracted and standardized from Jupyter notebooks to support both supervised finetuning and value-based learning. Building on this, we propose JUPITER, a framework that formulates data analysis as inference-time value-guided search, enabling more accurate and efficient multi-step reasoning under limited interaction steps. Experimental results on InfiAgent-DABench and other benchmarks show that our approach significantly boosts the performance of open-source models, enabling them to match or exceed proprietary models such as GPT-4o. Together, NbQA and JUPITER offer a scalable and practical solution for empowering LLMs with advanced data analysis capabilities.

References

Baghizadeh, K. 2024. Titanic Dataset. https://www.kaggle.com/datasets/heptapod/titanic. Accessed: 2025-07-15.

Brandfonbrener, D.; Henniger, S.; Raja, S.; Prasad, T.; Loughridge, C.; Cassano, F.; Hu, S. R.; Yang, J.; Byrd, W. E.; Zinkov, R.; and Amin, N. 2024. VerMCTS: Synthesizing Multi-Step Programs using a Verifier, a Large Language Model, and Tree Search. arXiv:2402.08147.

Bytedance-Seed-Foundation-Code-Team; :; Cheng, Y.; Chen, J.; Chen, J.; Chen, L.; Chen, L.; Chen, W.; Chen, Z.; Geng, S.; Li, A.; Li, B.; Li, B.; Li, L.; Liu, B.; Liu, J.; Liu, K.; Liu, Q.; Liu, S.; Liu, S.; Liu, T.; Liu, T.; Liu, Y.; Long, R.; Mai, J.; Ning, G.; Peng, Z. Y.; Shen, K.; Su, J.; Su, J.; Sun, T.; Sun, Y.; Tao, Y.; Wang, G.; Wang, S.; Wang, X.; Wang, Y.; Wang, Z.; Xia, J.; Xiang, L.; Xiao, X.; Xiao, Y.; Xi, C.; Xin, S.; Xu, J.; Xu, S.; Yang, H.; Yang, J.; Yang, Y.; Yuan, J.; Zhang, J.; Zhang, Y.; Zhang, Y.; Zheng, S.; Zhu, H.; and Zhu, M. 2025. FullStack Bench: Evaluating LLMs as Full Stack Coders. arXiv:2412.00535.

Chen, G.; Liao, M.; Li, C.; and Fan, K. 2024. AlphaMath Almost Zero: Process Supervision without Process. arXiv:2405.03553.

Cheng, L.; Li, X.; and Bing, L. 2023. Is GPT-4 a Good Data Analyst? arXiv:2305.15038.

De Bie, T.; De Raedt, L.; Hernández-Orallo, J.; Hoos, H. H.; Smyth, P.; and Williams, C. K. I. 2022. Automating data science. *Communications of the ACM*, 65(3): 76–87.

Donoho, D. L. 2017. 50 Years of Data Science. *Journal of Computational and Graphical Statistics*, 26: 745 – 766.

Hollmann, N.; Müller, S.; and Hutter, F. 2023. Large Language Models for Automated Data Science: Introducing CAAFE for Context-Aware Automated Feature Engineering. arXiv:2305.03403.

Hong, S.; Lin, Y.; Liu, B.; Liu, B.; Wu, B.; Zhang, C.; Wei, C.; Li, D.; Chen, J.; Zhang, J.; Wang, J.; Zhang, L.; Zhang, L.; Yang, M.; Zhuge, M.; Guo, T.; Zhou, T.; Tao, W.; Tang, X.; Lu, X.; Zheng, X.; Liang, X.; Fei, Y.; Cheng, Y.; Gou, Z.; Xu, Z.; and Wu, C. 2024. Data Interpreter: An LLM Agent For Data Science. arXiv:2402.18679.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685.

Hu, X.; Zhao, Z.; Wei, S.; Chai, Z.; Ma, Q.; Wang, G.; Wang, X.; Su, J.; Xu, J.; Zhu, M.; Cheng, Y.; Yuan, J.; Li, J.; Kuang, K.; Yang, Y.; Yang, H.; and Wu, F. 2024. InfiAgent-DABench: Evaluating Agents on Data Analysis Tasks. arXiv:2401.05507.

Learning, U. M. 2024. Wine Quality Dataset. https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009. Accessed: 2025-07-15.

Mai, X.; Xu, H.; W, X.; Wang, W.; Hu, J.; Zhang, Y.; and Zhang, W. 2025. Agent RL Scaling Law: Agent RL with Spontaneous Code Execution for Mathematical Problem Solving. arXiv:2505.07773.

Manimala. 2024. Iris Dataset. https://www.kaggle.com/datasets/vikrishnan/iris-dataset/data. Accessed: 2025-07-15.

Namdari, R. 2024. Breast Cancer Dataset. https://www.kaggle.com/datasets/reihanenamdari/breast-cancer. Accessed: 2025-07-15.

OpenAI. 2025. OpenAI Platform: Code Interpreter Tool. https://platform.openai.com/docs/assistants/tools/code-interpreter. Accessed: 2025-07-23.

Qiao, B.; Li, L.; Zhang, X.; He, S.; Kang, Y.; Zhang, C.; Yang, F.; Dong, H.; Zhang, J.; Wang, L.; Ma, M.; Zhao, P.; Qin, S.; Qin, X.; Du, C.; Xu, Y.; Lin, Q.; Rajmohan, S.; and Zhang, D. 2024. TaskWeaver: A Code-First Agent Framework. arXiv:2311.17541.

Schirmer, C. 2024. Boston Housing Dataset. https://www.kaggle.com/datasets/schirmerchad/bostonhoustingmlnd. Accessed: 2025-07-15.

Shen, Y.; Song, K.; Tan, X.; Li, D.; Lu, W.; and Zhuang, Y. 2023. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. arXiv:2303.17580.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; and Zhou, D. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. arXiv:2203.11171.

Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; Awadallah, A. H.; White, R. W.; Burger, D.; and Wang, C. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155.

Yang, Z.; Zhou, Z.; Wang, S.; Cong, X.; Han, X.; Yan, Y.; Liu, Z.; Tan, Z.; Liu, P.; Yu, D.; Liu, Z.; Shi, X.; and Sun, M. 2024. MatPlotAgent: Method and Evaluation for LLM-Based Agentic Scientific Data Visualization. arXiv:2402.11453.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T. L.; Cao, Y.; and Narasimhan, K. 2023a. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2023b. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629.

You, Z.; Zhang, Y.; Xu, D.; Lou, Y.; Yan, Y.; Wang, W.; Zhang, H.; and Huang, Y. 2025. DatawiseAgent: A Notebook-Centric LLM Agent Framework for Automated Data Science. arXiv:2503.07044.

Zhang, M. R.; Desai, N.; Bae, J.; Lorraine, J.; and Ba, J. 2024. Using Large Language Models for Hyperparameter Optimization. arXiv:2312.04528.

Świechowski, M.; Godlewski, K.; Sawicki, B.; and Mańdziuk, J. 2022. Monte Carlo Tree Search: a review of recent modifications and applications. *Artificial Intelligence Review*, 56(3): 2497–2562.