Untangling Knots: Leveraging LLM for Error Resolution in Computational Notebooks

KONSTANTIN GROTOV*, JetBrains Research, Germany
SERGEY TITOV*, JetBrains Research, Cyprus
YAROSLAV ZHAROV, JetBrains Research, Germany
TIMOFEY BRYKSIN, JetBrains Research, Cyprus

Computational notebooks became indispensable tools for research-related development, offering unprecedented interactivity and flexibility in the development process. However, these benefits come at the cost of reproducibility and an increased potential for bugs. There are many tools for bug fixing; however, they are generally targeted at the classical linear code. With the rise of code-fluent Large Language Models, a new stream of smart bug-fixing tools has emerged. However, the applicability of those tools is still problematic for non-linear computational notebooks. In this paper, we propose a potential solution for resolving errors in computational notebooks via an iterative LLM-based agent. We discuss the questions raised by this approach and share a novel dataset of computational notebooks containing bugs to facilitate the research of the proposed approach.

CCS Concepts: • **Computing methodologies → Planning for deterministic actions**.

Additional Key Words and Phrases: Computational notebooks, Agents, Planning

1 INTRODUCTION

Computational notebooks have become a popular medium for development during the last decade, especially for data analysis, machine learning [18], and creating educational [1], or scientific content [19]. One of the main features of computational notebooks is the possibility of a non-linear development process enabled by creating and executing cells with source code in any order. While this feature allows one to efficiently go through hypotheses and experiment a lot [24], it causes high code entanglement [22, 24] and, thus, a higher amount of errors in code. As a result, notebooks are struggling with low reproducibility rates [20, 21, 30], when notebooks hardly show the same results after re-running or even can not be executed without errors. And on the other hand, the notebooks' code often contains many stylistic errors [8].

In the previous works, several approaches were suggested to solve these issues. To mitigate the code entanglement problem, an algorithm was proposed for the preventive linearization [15]. While existing linters improve code quality, they are not adapted for notebooks and usually produce false positive errors. Hence, in the recent studies, additional notebook-specific linters and guidelines [25] and tools to match them were presented. Also, special versioning systems [2] were proposed to solve the reproducibility problem. The emergence of coding Large Language Models [12, 14, 17, 23] (LLMs) opens new prospects in this direction since these models can dynamically analyze the context, and address errors in a more nuanced way, compared to static analysis tools.

LLMs demonstrated decent bug-fixing abilities in common coding settings — repositories with regular scripts code in separate files [13]. However, there is not as much work on applying these techniques to the format of notebooks [16]. We suggest that solving the bug-fixing problem in the context of notebooks will benefit both fields — LLMs and computational notebooks. For the former, notebooks are an example of an interactive environment like works by Fan et al. [5], Goss et al. [7] to test their planning and execution abilities. In turn, LLMs could solve the most crucial problems for notebooks: code quality and reproducibility.

*Authors contributed equally to this research.

In this work, we take the first steps toward such a solution. Namely, we

(1) collect and share with the community a dataset [9] of Python projects that contain computational notebooks with at least one thrown exception;
(2) outline the agent-based [4, 26] approach for LLM usage in the notebooks and discuss the research questions we deem important for the practical applicability of this approach, as well as the way to estimate the quality of the solution.

2 DATASET OF NOTEBOOK ERRORS

Since the rise of the popularity of LLMs, they have already been used as error-solving tools [32, 33]. However, they were not applied to computational notebooks. To start filling that gap, we are sharing the dataset of computational notebooks containing errors [9]. The dataset contains 10,000 notebooks collected by taking a subset from a bigger unpublished corpus. The full corpus consists of all repositories with 10 or more stars on GitHub and all repositories containing Kotlin files. Before further filtering, we excluded forks and no longer actively maintained (archived) repositories to keep the variety of code in the dataset; we also removed the repositories with the last commit date before 2020 to have as many of the modern errors as possible and since popular datasets from Grotov et al. [8] and Pimentel et al. [20] provide a good subset of older notebooks. To gather the notebooks, we took the repositories with Python or Jupyter marked as primary languages, collected all notebooks containing errors, and randomly took the first 10,000 notebooks out of 12,609 from 4,636 repositories. We classified a notebook as containing errors if any of the cells' outputs resulted in a Python exception. The detection of these errors was facilitated by using the `output_type` property found within the JSON source of the notebook cells. The GitHub snapshot was accurate as of February 21, 2024. All the repositories in the dataset have a permissive license for future research (MIT, Apache-2.0, BSD-3-Clause) [6].

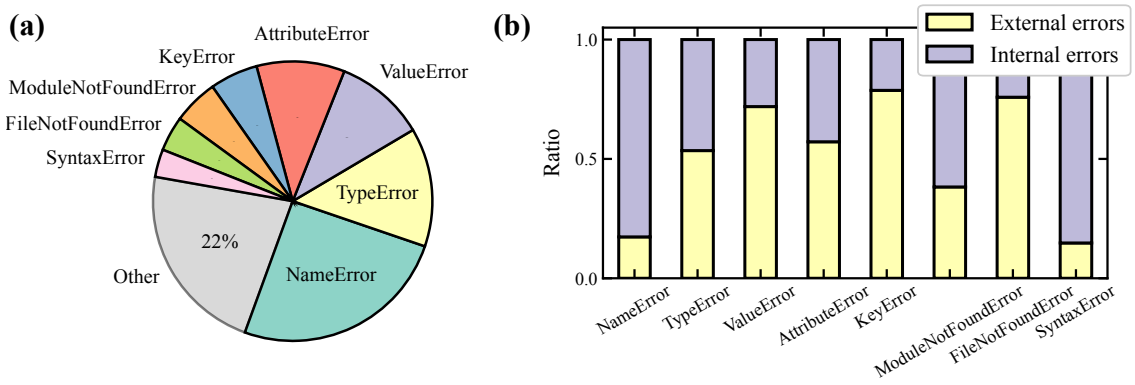

Fig. 1. (a) Distribution of top-8 most common errors in GitHub notebooks. (b) The ratio of internal and external errors for every of the top-8 error types.

To that end, we analyzed the distribution of the most common error types in the collected dataset. We selected all cells that contain errors in the output. Then, for every error, we determined the location of the code that raised the exception; we denote an error as internal if this location is within the notebook and as external otherwise. We determined every error's type using the default Python exception notation. Even though the dataset presents various error types (354 different types), most of the errors belong to a few classes. As illustrated in Figure 1 (a), the distribution

of diverse error types aligns with previous studies on notebook reproducibility [20]. Notably, most errors (78% of all errors found in the dataset) fall into eight categories. It is worth mentioning that the rest of the categories are sparse, and 282 of them contain less than 5 errors each. However, some of these are pretty narrow and understandable, such as `SyntaxError` or `ModuleNotFoundError`, while the top four error types are pretty general and could appear in various circumstances. These top-4 errors can be linked to diverse low-level issues commonly encountered in the non-linear development process [21] used in notebooks, as well as the loss of focus associated with it. For instance, a `NameError` could occur when the execution order becomes non-linear, and cells in the notebook are used before they are initialized. Furthermore, `TypeError` and `ValueError` errors might arise due to incorrect modifications of objects when the execution reverts back to a linear order. For instance, these errors could arise from the use of temporary variables that have the same name and are reassigned to different parts of the notebook. Also, our visual analysis of randomly chosen errors shows that many errors result from incorrect usage of packages and methods or their incompatibility with the installed environment. It's also important to mention that, as shown in Figure 1 (b), most of the mentioned errors are external (caused by the incorrect behavior outside of the notebook, for example, in an imported package), so the gathering additional context is valuable for solving these errors. To facilitate further research, we have made the dataset publicly available and shared the analysis code on Hugging Face [9].

3 FUTURE WORK

In this section, we briefly outline how to solve such a problem using the approach of a developing field of AI agents. Following this, we pose the research questions that we consider vital for the success of this approach.

3.1 Error Solving using LLM Agents

The field of LLM-based AI agents gained traction recently. Such agents have shown abilities to engage with software engineering tasks [27, 28], interact with web environments [3, 34], or operate an embodied agent [31]. We suggest employing such agents for iterative error resolution in computational notebooks. The core concept is to allow the agent to code and execute cells, utilizing the notebook's natural interactivity. The agent, therefore, will be able to expand and refine its context before solving the error.

Expanding the context by utilizing methods akin to ones that are used for regular code [29], such as incorporating notebook feedback on the proposed resolution or drawing from data sources and the notebook itself, can indeed be insightful. However, the interactive nature of notebooks might offer additional value.

The AI agent we envision gains control only after an error occurs and the user seeks a solution. This approach allows the agent to investigate the environment independently and will not destructively interfere with the developers' workflow. As mentioned, this boundary-free exploration is achievable by creating and executing temporary cells and processing the feedback. This provides an avenue for yielding supplemental contextual information which cannot be explicitly present.

3.2 Research Questions

We pose several research questions for ourselves and the community to solve. We think solving these questions will streamline the further development of LLM-based AI agents for the computational notebooks' bugfixing.

RQ1: *How to develop a secure playground?* The security aspect is vital as the environment should provide interaction and error-solving from the agent side, as well as reversibility to the initial user's codebase. We are considering various ways to implement this, from duplicating a notebook and running it in a sandbox to railguard the agent's actions.

RQ2: *Which metrics should be used to specifically evaluate an agent's performance in a computational notebook environment?* While we can certainly apply existing metrics to make comparisons among agents within a notebook, these measures may not always provide a meaningful view of performance given the context of a notebook environment. More specifically, even if a cell shows no errors, it is possible that the outcome may not align with the developer's original goal due to contextual differences.

RQ3: *What tools can (or should) an agent utilize in a computational notebook?* The expansion of the toolset increases the length of the context window occupied by the tool description. Since agents are, naturally, iterative, the context window can quickly be overflown by the dialogue. This leads to a desire to find a minimal yet complete set of tools. While we mentioned several possible tools for extending the context behind the exception message, we hypothesize that it is beneficial to study the impact of different tools on the agent performance.

RQ4: *Is it possible to accurately solve errors in computational notebooks using Open-Source LLMs?* Open models have already shown decent results for correcting errors in code [32], but their performance on computational notebooks has not yet been studied. Transitioning to open models could allow one to solve errors without the need for code sharing with third parties. Moreover, it will allow fine-tuning according to the relevant context.

RQ5: *Is the interaction between agents useful for error solving in computational notebooks?* One of the questions open to discussion is whether AI agents perform better as standalone error solvers or when interacting with other agents. Due to the potential for productive communication among agents possessing diverse specific features, we posit that these agents may yield more precise solutions [11]. Additionally, interaction with the user may be helpful [10].

4 CONCLUSION

In this paper, we considered the task of resolving errors in computational notebooks. We discussed the possibility of solving such issues with the interactive LLM-based agents. To facilitate further research, we stated questions that we consider essential for future developments. Alongside this, we collected, published, and analyzed a dataset of notebooks containing errors. We hypothesize that solving errors in computational notebooks with agents can be beneficial for research both on AI agents and human-notebook interaction.

REFERENCES

[1] Lorena A Barba, Lecia J Barker, Douglas S Blank, Jed Brown, Allen B Downey, Timothy George, Lindsey J Heagy, Kyle T Mandli, Jason K Moore, David Lippert, et al. 2019. Teaching and learning with Jupyter. *Recuperado: https://jupyter4edu. github. io/jupyter-edu-book* (2019), 1–77.

[2] Nachiket Deo, Boris Glavic, and Oliver Kennedy. 2022. Runtime provenance refinement for notebooks. In *Proceedings of the 14th International Workshop on the Theory and Practice of Provenance*. 1–4.

[3] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. WorkArena: How Capable Are Web Agents at Solving Common Knowledge Work Tasks? arXiv:2403.07718 [cs.LG]

[4] Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Yejin Choi, et al. 2024. Agent ai: Surveying the horizons of multimodal interaction. *arXiv preprint arXiv:2401.03568* (2024).

[5] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. https://openreview.net/forum?id=rc8o_j8I8PX

[6] Yaroslav Golubev, Maria Eliseeva, Nikita Povarov, and Timofey Bryksin. 2020. A study of potential code borrowing and license violations in java projects on github. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 54–64.

[7] Stephen A. Goss, Robert J. Steininger, Dhruv Narayanan, Daniel V. Olivença, Yutong Sun, Peng Qiu, Jim Amato, Eberhard O. Voit, Walter E. Voit, and Eric J. Kildebeck. 2023. Polycraft World AI Lab (PAL): An Extensible Platform for Evaluating Artificial Intelligence Agents. arXiv:2301.11891 [cs.AI]

[8] Konstantin Grotov, Sergey Titov, Vladimir Sotnikov, Yaroslav Golubev, and Timofey Bryksin. 2022. A large-scale comparison of Python code in Jupyter notebooks and scripts. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 353–364.

[9] Konstantin Grotov, Sergey Titov, Yaroslav Zharov, and Timofey Bryksin. 2023. Dataset of Errors in Jupyter Notebooks. https://huggingface.co/datasets/JetBrains-Research/jupyter-errors-dataset.

[10] Meiqi Guo, Mingda Zhang, Siva Reddy, and Malihe Alikhani. 2021. Abg-CoQA: Clarifying Ambiguity in Conversational Question Answering. In *3rd Conference on Automated Knowledge Base Construction*. https://openreview.net/forum?id=SlDZ1o8FsJU

[11] Shariq Iqbal and Fei Sha. 2019. Actor-attention-critic for multi-agent reinforcement learning. In *International conference on machine learning*. PMLR, 2961–2970.

[12] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[13] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. 2023. InferFix: End-to-End Program Repair with LLMs. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (, San Francisco, CA, USA,) *(ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1646–1656. https://doi.org/10.1145/3611643.3613892

[14] Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463* (2023).

[15] Stephen Macke, Hongpu Gong, Doris Jung-Lin Lee, Andrew Head, Doris Xin, and Aditya Parameswaran. 2020. Fine-grained lineage for safer notebook interactions. *arXiv preprint arXiv:2012.06981* (2020).

[16] Andrew M McNutt, Chenglong Wang, Robert A Deline, and Steven M Drucker. 2023. On the design of ai-powered code assistants for notebooks. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–16.

[17] OpenAI. 2023. GPT-4 Technical Report. *ArXiv* abs/2303.08774 (2023). https://arxiv.org/abs/2303.08774

[18] Jeffrey M Perkel. 2018. Why Jupyter is data scientists' computational notebook of choice. *Nature* 563, 7732 (2018), 145–147.

[19] Jeffrey M Perkel. 2021. Ten computer codes that transformed science. *Nature* 589, 7842 (2021), 344–349.

[20] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2019. A large-scale study about quality and reproducibility of jupyter notebooks. In *2019 IEEE/ACM 16th international conference on mining software repositories (MSR)*. IEEE, 507–517.

[21] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2021. Understanding and improving the quality and reproducibility of Jupyter notebooks. *Empirical Software Engineering* 26, 4 (2021), 65.

[22] Dhivyabharathi Ramasamy, Cristina Sarasua, Alberto Bacchelli, and Abraham Bernstein. 2023. Workflow analysis of data science code in public GitHub repositories. *Empirical Software Engineering* 28, 1 (2023), 7.

[23] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[24] Adam Rule, Aurélien Tabard, and James D Hollan. 2018. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[25] Sheeba Samuel, Frank Löffler, and Birgitta König-Ries. 2020. Machine learning pipelines: provenance, reproducibility and FAIR data principles. In *International Provenance and Annotation Workshop*. Springer, 226–230.

[26] Yonadav Shavit, Sandhini Agarwal, Miles Brundage, Steven Adler, Cullen O'Keefe, Rosie Campbell, Teddy Lee, Pamela Mishkin, Tyna Eloundou, Alan Hickey, et al. 2023. *Practices for Governing Agentic AI Systems*. Technical Report. OpenAI Technical Report. https://cdn. openai. com/papers/practices-for ….

[27] Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2024. Design2Code: How Far Are We From Automating Front-End Engineering? arXiv:2403.03163 [cs.CL]

[28] Michele Tufano, Anisha Agarwal, Jinu Jang, Roshanak Zilouchian Moghaddam, and Neel Sundaresan. 2024. AutoDev: Automated AI-Driven Development. arXiv:2403.08299 [cs.SE]

[29] Hanbin Wang, Zhenghao Liu, Shuo Wang, Ganqu Cui, Ning Ding, Zhiyuan Liu, and Ge Yu. 2023. INTERVENOR: Prompt the Coding Ability of Large Language Models with the Interactive Chain of Repairing. *arXiv preprint arXiv:2311.09868* (2023).

[30] Jiawei Wang, Li Li, and Andreas Zeller. 2021. Restoring execution environments of Jupyter notebooks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1622–1633.

[31] Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023. Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents. arXiv:2302.01560 [cs.AI]

[32] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE 2023). Association for Computing Machinery*.

[33] Kechi Zhang, Zhuo Li, Jia Li, Ge Li, and Zhi Jin. 2023. Self-Edit: Fault-Aware Code Editor for Code Generation. *arXiv preprint arXiv:2305.04087* (2023).

[34] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2023. WebArena: A Realistic Web Environment for Building Autonomous Agents. arXiv:2307.13854 [cs.AI]