It's LIT!
Reliability-Optimized LLMs with Inspectable Tools

Ruixin Zhang
Department of Computer Science
Duke University
ruixin.zhang@duke.edu

Jon Donnelly
Department of Computer Science
Duke University
jon.donnelly@duke.edu

Zhicheng Guo
Department of Computer Science
Duke University
zhicheng.guo@duke.edu

Ghazal Khalighinejad
Department of Computer Science
Duke University
ghazal.khalighinejad@duke.edu

Haiyang Huang
Department of Computer Science
Duke University
haiyang.huang@duke.edu

Alina Jade Barnett
Department of Computer Science
Duke University
alina.barnett@duke.edu

Cynthia Rudin
Department of Computer Science
Duke University
cynthia@cs.duke.edu

Abstract

Large language models (LLMs) have exhibited remarkable capabilities across various domains. The ability to call external tools further expands their capability to handle real-world tasks. However, LLMs often follow an opaque reasoning process, which limits their usefulness in high-stakes domains where solutions need to be trustworthy to end users. LLMs can choose solutions that are unreliable and difficult to troubleshoot, even if better options are available. We address this issue by forcing LLMs to use external – more reliable – tools to solve problems when possible. We present a framework built on the tool-calling capabilities of existing LLMs to enable them to select the most reliable and easy-to-troubleshoot solution path, which may involve multiple sequential tool calls. We refer to this framework as LIT (LLMs with Inspectable Tools). In order to support LIT, we introduce a new and challenging benchmark dataset of 1,300 questions and a customizable set of reliability cost functions associated with a collection of specialized tools. These cost functions summarize how reliable each tool is and how easy it is to troubleshoot. For instance, a calculator is reliable across domains, whereas a linear prediction model is not reliable if there is distribution shift, but it is easy to troubleshoot. A tool that constructs a random forest is neither reliable nor easy to troubleshoot. These tools interact with the Harvard USPTO Patent Dataset and a new dataset of NeurIPS 2023 papers to solve mathematical, coding, and modeling problems of varying difficulty levels. We demonstrate that LLMs can achieve more

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Workshop on Multi-Turn Interactions in Large Language Models.

reliable and informed problem-solving while maintaining task performance using our framework.

1 Introduction

Large language models (LLMs) are arguably the most complicated black box machine learning algorithms currently in existence. Their transformer architectures make them extremely difficult to trust and troubleshoot. When they answer difficult questions wrong, no one knows why, and no one can figure out why [6]. One path towards a potential solution is the use of *external tools* by LLMs [20]. Rather than depending on a single huge LLM to solve every problem or perform every calculation, external tools specialize in various tasks and can be called by the LLM to perform them as part of solving a larger problem. For instance, rather than depending on a single LLM to solve (502*2952+323)/63, the LLM could call a calculator. Other tools might retrieve datasets, execute code, or perform other tasks. Allowing LLMs to call external tools offers a promising way to mitigate their complexity for several reasons. First, tools are specialized and more reliable; second, using them makes the LLM's logic more modular, and allows developers to take responsibility for key steps in forming a solution [2, 7, 9, 20, 21, 33].

Importantly, not all tools are created the same. They vary tremendously in their reliability and ease of troubleshooting. For instance, while a calculator is completely reliable given the correct inputs, a specialized model for solving physics problems may not be (though it might be more reliable for physics problems than the more general LLM calling it, and certainly easier to troubleshoot). Prior work on tool-based LLMs [18, 31] failed to account for the relative reliability and inspectability of tools within their toolboxes, which means their LLMs' solutions are likely more opaque and difficult to troubleshoot than necessary. Prior to the current work, there existed no metrics, baseline methods, standard frameworks, or question banks for exploring the reliability of tool selection in LLMs.

We address this shortcoming by explicitly encouraging an LLM to prefer reliable and inspectable tools over alternatives by assigning a reliability and inspectability cost to each tool call and choosing the solution with the minimum cost. This allows the LLM to provide reliable and inspectable reasoning whenever possible without sacrificing the ability to answer difficult prompts correctly. This is illustrated in Figure 1. For example, imagine that we asked an LLM which five authors have the most NeurIPS 2023 papers with titles that mention "Large Language Models." The LLM might simply provide a random list of 5 author names, but we would prefer that it use SQL code with reliable logic to identify these 5 authors.

We introduce the LLMs with Inspectable Tools (LIT) framework. During inference, the LLM generates multiple candidate solutions, each consisting of a sequence of tool calls. For each solution, the model computes the total cost based on the individual costs of each tool that it uses. It then selects the most reliable and inspectable option that still ensures accurate results. Once the optimal solution is identified, the LLM sequentially executes the necessary tool calls to produce the final answer.

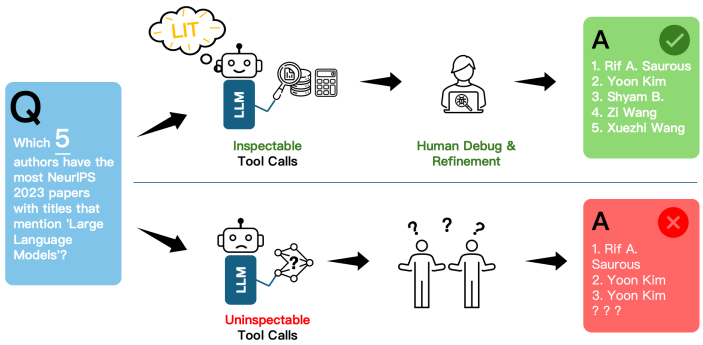

Figure 1: Conceptual summary. By using the LIT framework, the LLM selects more reliable and inspectable tools, allowing human users to debug and refine the solution proposed by the LLM, producing the correct result. In contrast, the vanilla LLM without LIT selects uninspectable solutions with reasoning that cannot be inspected or corrected.

We evaluate this framework using a suite of specialized tools with varying levels of reliability and inspectability, a novel benchmark dataset of 1,300 questions with varying levels of difficulty, and two distinct problem domains. We use 50% of these questions for validation, and the remaining 50% for testing (the LIT framework does not include any model training). We find that prompting the LLM to prefer reliable and inspectable tools dramatically increases the transparency of the given solutions, and also improves the model's accuracy in answering most of these questions. In summary, we provide the following contributions:

1. We introduce a prompting framework that explicitly accounts for differences in the reliability and inspectability of tools (Appendix C contains detailed prompts). We show that, across several LLMs, this framework leads to responses that exhibit enhanced inspectability while maintaining accuracy.
2. We provide a suite of 8 tools with varying levels of reliability and inspectability, including tools for arithmetic operations (Calculator), dataset retrieval (DBLoader), flexible code execution (PandasInterpreter, PythonInterpreter), time series forecasting (Forecaster), text classification (TextualClassifier), language model inference (LLMInferencer), and result consolidation (Finish).[1]
3. We introduce a new benchmark for tool-based LLMs consisting of 1,300 questions about two external databases: the Harvard USPTO Patent Dataset [24] and a novel database with metadata from NeurIPS papers from 2023.

2 Related work

2.1 Tool learning

Our work is closely related to the topic of tool learning. Tool learning allows LLMs to use external tools to solve complex problems, gain real-time domain knowledge, or mitigate their own limitations [9, 10, 13, 16, 25, 29, 30, 33]. Although previous work has shown the effectiveness of tool learning, if there are multiple choices of viable tools available, the algorithm will somehow choose one set of tools (or choose to call none at all), and the decision logic would likely remain opaque to human operators. In previous work on both single-step problem solving [12, 20, 21] and iterative multi-step tool calling with feedback [8, 11, 17, 23], the LLM is optimized *only* for task success and solution correctness, while in reality, optimizing for other objectives, such as reliability or the ability to troubleshoot, may yield more useful results. In this work, we focus on reliability and inspectable tool usage, which is compatible with any of the above tool-calling paradigms. Our experiments examine the case of sequential, dependent tool calling.

2.2 Tool learning datasets and benchmarks

There has been a rapid development of benchmark datasets for tool learning with LLMs [11, 15, 22, 23, 25, 32]. Existing benchmark datasets largely focus on public API calls as tools, and evaluations mainly focus on the call counts and metrics based on task success (e.g., precision, F-1 score, rank). Our dataset provides a much-needed supplement to existing benchmarks, allowing the manual creation of additional tools and customizable reliability metrics in addition to traditional success rate-based metrics. Its built-in tools range from custom functions to external models. The questions in our benchmark dataset have varying degrees of difficulty, enabling researchers to measure the ability of LLMs to select reliable and inspectable tools.

3 Framework

Given a set of tools, LIT consists of two key components: a set of tool-cost functions, and a carefully designed few-shot prompt to guide the underlying LLM to prefer more reliable and inspectable – or, lower cost – tools. We first provide the general principles for defining tool-cost functions, then introduce the prompt strategy used in LIT.

[1] Each tool is designed to offer deterministic results for reproducibility.

3.1 Tool cost design

The cost of a solution is the combined costs of all the tools used in the solution sequence. Drawing inspiration from human-computer interaction (HCI) literature, which emphasizes reliability, debuggability, and simplicity as desirable qualities of systems [19], we consider three primary criteria when determining the cost of a tool: whether the tool achieves robust performance across inputs, the ease of debugging the tool, and the complexity of the arguments to the tool. These costs can be determined differently by the user in each context: while a calculator may be reliable across domains, a specialized LLM model may be less reliable when used out of its training distribution. We present a list of reasonable values in Table 3, applying each criterion to the tools considered, although these values can easily be adjusted and set according to the problem context. Each criterion is described in detail below:

Costs should consider how robust performance is across inputs (P). We prefer tools that, given some input, reliably produce exactly the output a user expects. For example, tools like DBLoader and the Calculator demonstrate robust performance because DBLoader simply loads a dataset and any reasonable calculator always produces correct results for arithmetic operations. In contrast, model-based tools exhibit less robust performance. For example, the Forecaster (based on an AutoRegressive Integrated Moving Average – ARIMA – model) relies on the stationarity and linearity of the data, sufficient data points, and the presence of autocorrelation [3]. ARIMA's predictions are unreliable without these conditions.

Costs should consider ease of debugging (D). We consider tools that can easily be debugged. For example, tools like the PandasInterpreter and PythonInterpreter are designed to accept code as their arguments. This trait makes them easier to debug, as modifying the input arguments to address errors or adjust behavior is straightforward, since the mapping between a given input argument and the tool's output is completely transparent. In contrast, tools like the Forecaster and TextualClassifier have more rigid argument structures. A Forecaster typically requires previous time series data and a forecast length as its primary inputs, while a TextualClassifier calls a trained model for predicting a binary outcome, which could be either a BERT model [5], a logistic regression model, or a convolutional neural network model. The LLM decides which arguments to put in, what inputs to use, which model type to use, and which target variable to predict.

The limited number of argument combinations available for these tools reduces their flexibility. It can also make troubleshooting more difficult, as potential sources of errors are harder to modify. Moreover, it is not always clear how changing one parameter of a predictive model will change its predictions, particularly for complex predictive models.

Costs should consider the complexity of arguments (C). We consider tools to be more reliable and easier to troubleshoot when they take simple arguments. The susceptibility of tools to errors often correlates with the complexity of the arguments they handle. For instance, tools like the PandasInterpreter and PythonInterpreter become more prone to errors as the complexity of the input code increases. Factors such as additional lines of code or the inclusion of multiple imported packages introduce more dependencies and interactions, raising the likelihood of issues [28]. Similarly, in the case of the TextualClassifier, the choice of trained model significantly influences error susceptibility. More complex models, such as BERT, involve additional parameters and intricate processes, introducing additional potential points of failure compared to simpler models like logistic regression. Consequently, tools handling more complex arguments incur higher costs due to the increased risk of execution issues.

3.2 Reliability prompt and LLM reasoning

We now introduce the prompt strategy used in LIT. We designed a prompt (described in detail in Appendix C) that incorporates customizable cost formulas for each tool, as described in the previous section, along with 5 examples of reliable and inspectable solutions for questions not included in our benchmark dataset. Within this prompt, the LLM is instructed to generate as many solutions as possible, up to a maximum of four, to avoid duplication and manage token usage efficiently. Each solution involves sequential tool calls. The LLM independently generates reasoning chains, which include calculating the overall cost for each solution. It then compares these costs to select the most reliable/inspectable solution while ensuring that accuracy is not compromised. Part of the LLM reasoning is also to decide how long each sequence of tool calls should be. The sequence is ended

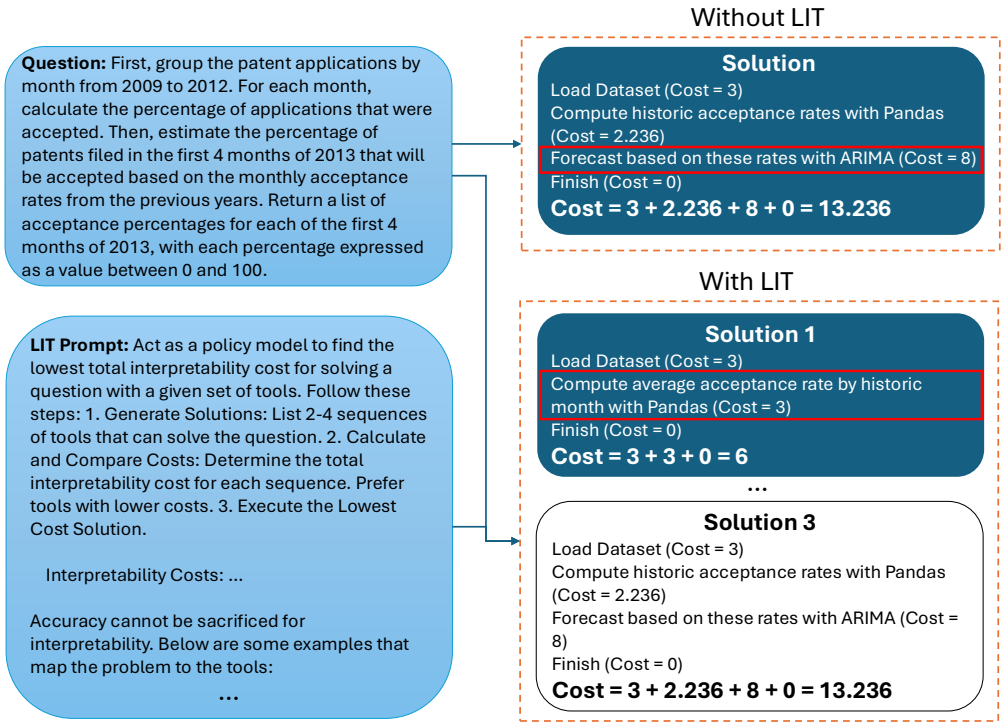

Figure 2: A simplified example of the prompting framework used by LIT. In LIT, we provide the model a cost for each tool and instruct the model to provide multiple alternative solutions, selecting the one with the lowest cost when possible.

by calling the tool "Finish." Once the optimal solution is chosen, the LLM proceeds to execute it, making tool calls sequentially. Figure 2 provides a simple example of this process.

4 Benchmark details

Let us describe the new reliability/inspectability benchmark. We created a set of 1,300 questions concerning one existing and one new dataset, along with a suite of 8 tools that each LLM can leverage.

4.1 Datasets

Harvard USPTO Patent Dataset (HUPD) The HUPD dataset [24] comprises 4.5 million patent applications filed with the United States Patent and Trademark Office (USPTO) between 2004 and 2018. Each patent application is structured as a JSON file that contains the abstract, background, claims, summary, and full description. In addition to the text fields, the JSON file includes metadata such as the inventors' names, cities, filing dates, and the USPTO's final decision.

The abundance of metadata allows us to use various tools that can be called by an LLM. Our questions can be answered with either reliable/inspectable or unreliable/uninspectable tools. For example, we ask the LLM to estimate the average filing time within a certain time frame, or to predict the likelihood of an application being accepted or not. These problems could be handled by the LLM alone, or by calling tools that make database queries, calling tools that constrain the selection of the database to given values, tools that create datasets for logistic regression and then run it, etc.

NeurIPS 2023 Papers Dataset Using web scraping techniques, we collected information for each accepted NeurIPS 2023 paper, including the title, author list, abstract, topic, and whether the paper was accepted for an oral presentation. For this dataset, we might analyze the distribution of the number of authors or estimate the likelihood of a paper being selected for an oral presentation.

4.2 Tools

We developed a suite of 8 tools with varying levels of reliability/inspectability for the LIT framework, described here and in Appendix E. This suite ranges from simple, reliable tools such as a basic calculator and a database loader to more complex, uninspectable modeling tools that employ BERT-based and convolutional neural network models. These tools enable LLMs to effectively address queries related to each dataset. In our experiments, we consider sequential, dependent tool calling.

- Calculator: This tool performs arithmetic operations based on input expressions containing only numbers and operators (e.g., 2×3).
- DBLoader: This tool loads a specified database (either the HUPD or the NeurIPS 2023 Papers Dataset) as indicated by the DBName parameter. It also allows filtering of the dataset by specifying a subset, which can include either years or specific rows of the dataframe.
- PandasInterpreter: This tool interprets Pandas code written in Python, utilizing the database stored in the variable *df*, and returns a dictionary containing the values of variables defined within that code. It is particularly useful when the inquiry necessitates data manipulation on structured databases.
- PythonInterpreter: This tool interprets Python code and returns a dictionary containing the values of variables defined within that code.
- Forecaster: This tool executes a specified forecasting model (either an ARIMA model or a linear regression model) on historical data to predict subsequent data points (e.g., the patent acceptance rates over the next three months).
- TextualClassifier: This tool applies a trained binary classification model (either BERT, logistic regression, or convolutional neural network) to an input string to make predictions. The model, input, and target variable must be specified.
- LLMInferencer: This tool leverages the current LLM to generate solutions only when answers cannot be determined through other tools or when the queries involve complex reasoning.
- Finish: This tool concludes the task and returns the final answer, ensuring that all variable values are derived directly from the output of the previous tool call and are of the correct return data type.

4.3 Questions

The questions are generated from carefully designed templates that require the use of multiple tools in sequence to solve the problem. No problem can be addressed with a single tool. We define 13 question templates, provided in Appendix D, which are used to produce many distinct questions by filling in different specific values. For example, the question template "How does the number of patent applications filed in {year1} compare proportionally to those filed in {year2}?" may produce questions like "How does the number of patent applications filed in 2019 compare proportionally to those filed in 2020?" or "How does the number of patent applications filed in 2018 compare proportionally to those filed in 2020?" Each question has multiple possible solution approaches and a singular answer, so that under prompts specifying costs, the LLM generates several solution sets and selects the one with the lowest cost.

All questions require accessing information from the provided databases and are classified into three categories: easy, medium, and hard. Easy questions are designed to be optimally solved using reliable/inspectable tools, e.g., "How does the number of patent applications filed in {year1} compare proportionally to those filed in {year2}?" Medium questions can be solved effectively using either reliable/inspectable or unreliable/uninspectable tools, with comparable performance, e.g., "For a patent application not present in the database, with an abstract {abstract_content}, predict whether it will be accepted. Return either 'ACCEPTED' or 'NOT ACCEPTED."' Hard questions are structured to be best addressed by black box tools, e.g., "Predict the best-fit topic for the title of a NeurIPS paper, not present in the database: {title}. Options: {topic1}, {topic2}, {topic3}."

5 Experimental results

In this section, we evaluate the empirical efficacy of LIT. We consider 5 different LLMs with and without the LIT framework: GPT-3.5-Turbo [4], GPT-4-Turbo [14], Gemini-1.5-Pro-001 [26], Claude-3.5-Sonnet-Latest [1], and Meta-Llama-3.1-70B-Instruct-Turbo [27]. We evaluate each model across all 13 question formats included in our benchmark, requiring access to each of the 8 tools and the 2 datasets included in the benchmark. Details of the evaluation metrics can be found in Appendix A. Importantly, we always have access to the uninspectable baseline, so if LIT performs worse than it,

LLM	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	easy	med	hard
GPT-3.5	5.40	4.79	5.88	4.74	4.18	9.87	NA	18.70	30.00	15.67	30.00	30.00	24.86	5.81	17.32	28.29
GPT-3.5 LIT	4.53	4.05	6.21	4.61	4.75	4.27	10.90	9.17	NA	11.30	4.87	18.57	26.07	4.74	10.46	16.50
GPT-4	5.91	4.56	6.88	5.14	6.27	5.42	30.48	20.00	30.00	19.74	30.00	30.00	30.00	5.70	25.06	30.00
GPT-4 LIT	5.29	4.71	5.35	5.01	5.79	4.89	8.68	8.38	30.00	7.00	32.58	30.00	30.00	5.17	13.52	30.86
Gemini	4.00	4.66	5.87	4.12	4.65	4.24	16.68	19.90	30.00	28.11	30.00	30.00	30.00	4.59	23.67	30.00
Gemini LIT	4.00	5.90	4.05	4.00	4.52	3.91	4.11	17.33	30.00	11.58	30.00	30.00	30.00	4.40	15.76	30.00
Claude	5.46	5.17	5.43	6.11	6.90	5.07	9.69	20.00	31.47	20.00	30.06	30.00	30.12	5.52	20.29	30.06
Claude LIT	4.54	4.75	4.97	4.86	5.60	4.61	12.25	11.58	30.31	16.32	29.67	30.00	30.06	5.06	17.62	29.91
Llama-3.1	5.75	5.50	5.31	4.91	6.67	4.79	5.31	7.00	30.00	10.52	9.88	19.50	23.34	5.66	13.21	17.57
Llama-3.1 LIT	4.25	5.35	5.29	5.04	6.32	4.89	6.22	7.00	30.00	7.43	8.24	19.54	8.93	5.19	12.66	12.24

statistically significant reduction in reliability/inspectability cost with LIT

statistically significant increase in reliability/inspectability cost with LIT

no statistically significant difference in reliability/inspectability cost

unable to conduct statistical test

Table 1: Comparison of reliability/inspectability cost. For each question (columns 1-13) and different LLMs (rows), we compare the mean cost of our LIT method with that of the standard, uninspectable baseline using a one-sided independent two-sample t-test. Columns 14-16 contain the averages for "easy" (Q1-Q6), "medium" (Q7-Q10) and "hard" (Q11-Q13) questions.

LLM	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	easy	med	hard
GPT-3.5	0.95	0.83	0.78	0.56	0.06	0.39	NA	0.42	0.53	0.35	0.30	0.26	0.14	0.60	0.33	0.23
GPT-3.5 LIT	0.61	0.71	0.93	0.46	0.18	0.81	0.00	0.42	NA	0.56	0.18	0.17	0.21	0.62	0.25	0.19
GPT-4	0.93	0.90	0.96	1.00	0.92	0.16	0.00	0.43	0.67	0.31	0.28	0.85	0.54	0.81	0.35	0.56
GPT-4 LIT	0.75	0.99	0.96	1.00	0.90	0.81	0.00	0.41	0.25	0.56	0.30	0.94	0.50	0.90	0.31	0.58
Gemini	1.00	0.27	0.91	0.40	0.27	0.97	0.00	0.37	0.49	0.45	0.40	0.96	0.33	0.64	0.33	0.56
Gemini LIT	1.00	0.57	0.98	0.56	0.24	0.90	0.00	0.36	0.54	0.34	0.50	0.96	0.48	0.71	0.31	0.65
Claude	1.00	0.94	1.00	0.40	0.82	1.00	0.01	0.43	0.71	0.31	0.61	1.00	0.58	0.86	0.37	0.73
Claude LIT	1.00	0.92	1.00	1.00	0.78	1.00	0.02	0.42	0.66	0.31	0.61	1.00	0.56	0.95	0.35	0.72
Llama-3.1	0.75	0.36	1.00	0.55	0.52	0.67	0.00	0.21	0.68	0.24	0.06	0.36	0.24	0.64	0.28	0.22
Llama-3.1 LIT	1.00	0.37	1.00	0.37	0.05	0.66	0.00	0.28	0.48	0.30	0.00	0.64	0.19	0.58	0.27	0.28

statistically significant improved performance with LIT

statistically significant decreased performance with LIT

no statistically significant difference in performance

unable to conduct statistical test

Table 2: Comparison of performance (metrics are detailed in Appendix A as they differ depending on the type of answer required). For each question (columns 1-13) and different LLMs (rows) we compare our LIT method to the standard, black box baseline using a one-sided independent two-sample t-test. Columns 14-16 contain the averages for "easy" (Q1-Q6), "medium" (Q7-Q10) and "hard" (Q11-Q13) questions. The LIT framework performed well for "easy" and "hard" questions, but performed worse on "medium" difficulty questions.

we can always switch back to it. The question we investigate is whether LIT often has an advantage over the black box baseline.

LIT improves reliability/inspectability We evaluated the reliability/inspectability of LIT's answers by measuring the average cost (as specified in Table 3) of each model's solution across realizations of each question template. As shown in Table 1, we found that LIT produced a solution with similar or better reliability/inspectability in 61 out of 65 cases. Notably, LIT improved reliability/inspectability for most questions across all 5 LLM backbones we considered. Moreover, we find that LIT improved or matched the average cost for easy- and medium-difficulty questions in every case except one (namely, Q2). We found that this trend did not hold as strongly for hard questions because LLMs produce high costs both with and without LIT, suggesting that these questions cannot be easily solved using the available reliable/inspectable tools.

LIT maintains performance We evaluated the performance of LIT for each question template. As shown in Table 2, LIT resulted in improved or comparable performance in 48 out of 65 settings. That is, the gains in reliability and inspectability provided by LIT do not generally *trade off* with performance; instead, they *maintain or improve* performance.

LIT's responses are easier to debug Here, we present example responses from Claude-3.5-Sonnet-Latest with and without LIT. Figure 3 presents an illustrative case from Question 10, in which the model must predict whether a paper would be accepted for oral presentation at NeurIPS based on its abstract. In this case, the black-box solution uses the pre-trained BERT model within the

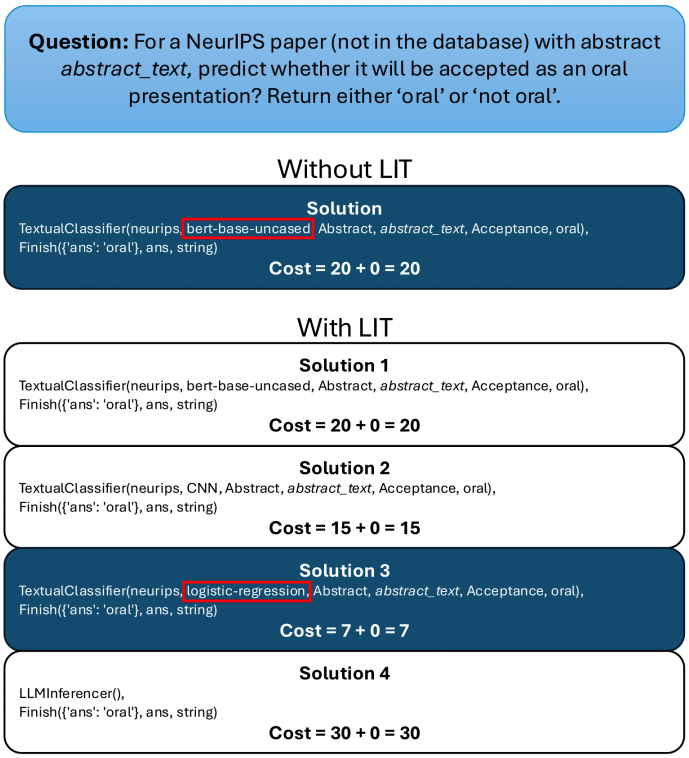

Figure 3: Solutions from an LLM to a question about whether a paper would be accepted to NeurIPS, with and without LIT. When using LIT, the model generates multiple candidate solutions and selects the one with the best cost. As a result, the model with LIT uses a logistic regression model rather than a BERT model to form its prediction, thereby using a substantially more inspectable tool. "Finish" tool signifies the end of logic stream and checks that the correct data type is returned. It has 0 cost.

TextualClassifier tool, while LIT's solution uses a pre-trained logistic regression model within this tool. The models achieve comparable accuracy, however, the logistic regression model offers a significant advantage. Its coefficients can be directly accessed and analyzed, providing clear insight into how it generates predictions, making it easy to troubleshoot. In contrast, the BERT model, with its vast number of parameters, has an inexplicable decision-making process. This is captured by the lower cost of 7 for the LIT solution compared to 20 for the black-box solution. Therefore, while both solutions are equally accurate, the LIT solution provides a distinct advantage for understanding and debugging the outputs. Appendix B provides an additional example.

Our benchmark is challenging While LIT demonstrates improved reliability/inspectability and comparable performance to unaltered LLMs, we found that a variety of questions in this benchmark are not well solved by *any* existing LLM techniques. As shown in Table 2, no model achieves strong performance on Question 7 (with R^2 scores close to 0, indicating a failure to explain the variability in the target variable and performance only marginally better than that of a naïve mean predictor), and performance is generally low for all medium and hard questions. This is particularly notable because medium-difficulty questions tend to be solvable using multiple different tools, meaning many different solutions with different reliability/inspectability costs are possible if an LLM can find them. As such, this represents a direct avenue for future work.

6 Conclusion

We introduced LIT, a framework for reliable and inspectable tool use for LLMs. In order to evaluate LIT, we introduced a novel benchmark of 1,300 questions and a suite of 8 tools accessing two distinct external datasets. We showed that, simply by applying LIT, we can steer a wide variety of

modern LLM models to select more reliable and inspectable tools without substantial cost to task performance.

LIT represents a substantial step towards more trustworthy LLM deployment. By leading LLMs to reason with inspectable tools, LIT provides a new level of transparency in LLM reasoning to users. Moreover, the benchmark introduced in this work will enable further work in this direction, ultimately leading to a more reliable future for LLMs.

We are aware of one primary limitation. The prompting strategy used in LIT increases both the amount of tokens given as input to each LLM and the amount of tokens generated as output (since the LLM must compare multiple solutions). This may pose a challenge when users are constrained by context limits. Future work should aim to improve the token efficiency of LIT.

References

[1] Anthropic. Claude. https://www.anthropic.com/, 2024. [Online].

[2] Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. Small language models are the future of agentic AI, 2025. URL https://arxiv.org/abs/2506.02153.

[3] G.E.P. Box, G.M. Jenkins, and G.C. Reinsel. *Time Series Analysis: Forecasting and Control*. Wiley, 2015.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[6] Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 2024.

[7] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *International Conference on Machine Learning*. PMLR, 2023.

[8] Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18030–18038, 2024.

[9] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*, 2023.

[10] Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. ToolkenGPT: Augmenting frozen language models with massive tools via tool embeddings. In *Advances in Neural Information Processing Systems*, volume 36, 2024.

[11] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Empirical Methods in Natural Language Processing*, 2023.

[12] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2024.

[13] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[14] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[15] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, 2024.

[16] Yujia Qin, Zihan Cai, Dian Jin, Lan Yan, Shihao Liang, Kunlun Zhu, Yankai Lin, Xu Han, Ning Ding, Huadong Wang, Ruobing Xie, Fanchao Qi, Maosong Sun Zhiyuan Liu, and Jie Zhou. WebCPM: Interactive web search for Chinese long-form question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023.

[17] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao1, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024.

[18] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19:198343, 2025.

[19] Muhammad Raees, Inge Meijerink, Ioanna Lykourentzou, Vassilis-Javed Khan, and Konstantinos Papangelis. From explainable to interactive AI: A literature review on current trends in human-AI interaction. *International Journal of Human-Computer Studies*, 189:103301, 2024.

[20] Timo Schick, Joanna Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

[21] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI tasks with ChatGPT and its friends in hugging face. In *Advances in Neural Information Processing Systems*, volume 36, 2024.

[22] Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation. In *Advances in Neural Information Processing Systems*, 2024.

[23] Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, Ye Tian, and Sujian Li. RestGPT: Connecting large language models with real-world applications via RESTful APIs. *arXiv preprint arXiv.2306.06624*, 2023.

[24] Mirac Suzgun, Luke Melas-Kyriazi, Suproteem Sarkar, Scott D Kominers, and Stuart Shieber. The harvard uspto patent dataset: A large-scale, well-structured, and multi-purpose corpus of patent applications. *Advances in Neural Information Processing Systems*, 36, 2024.

[25] Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023.

[26] Gemini Team. Gemini: A Family of Highly Capable Multimodal Models. *arXiv preprint arXiv:2312.11805*, 2023.

[27] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[28] Ana Trisovic, Matthew K. Lau, Thomas Pasquier, and Mercè Crosas. A large-scale study on research code quality and execution. *Scientific Data*, 9:60, 2022.

[29] Sai H Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. ChatGPT for robotics: Design principles and model abilities. *IEEE Access*, 2024.

[30] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *International Conference on Machine Learning*, 2024.

[31] Zhiruo Wang, Zhoujun Cheng, Hao Zhu, Daniel Fried, and Graham Neubig. What are tools anyway? a survey from the language model perspective. In *Conference on Language Modeling*, October 2024.

[32] Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the tool manipulation capability of open-source large language models. In *Foundation Models for Decision Making*, 2023.

[33] Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R Fung, Hao Peng, and Heng Ji. Craft: Customizing LLMs by creating and retrieving from specialized toolsets. In *International Conference on Learning Representations*, 2023.

A Performance Evaluation Details

We employ various methods to quantify performance depending on the nature of the questions posed. All performance values are between 0 and 1, facilitating straightforward interpretation and aggregation across easy, medium, and hard questions through averaging.

- **Q1, Q3, Q6:** These questions are evaluated based on correctness within a specified threshold. An answer provided by the language model (LLM) is deemed correct if the absolute difference between the LLM's answer and the ground truth answer is within 0.5% of the ground truth value. If this condition is met, the LLM answer receives a performance value of 1; otherwise, it receives a value of 0. This approach accounts for floating-point precision issues and minor discrepancies arising from variations in coding logic in the use of PandasInterpreter and PythonInterpreter.
- **Q2, Q5:** The performance for these questions is assessed through set intersection. The ground truth answers consist of lists of values, and performance is calculated as the number of overlapping elements between the LLM answer and the ground truth answer divided by the total number of elements in the ground truth list. This metric effectively captures the degree of agreement between the LLM's output and the expected results.
- **Q4:** Evaluation for this question involves a boolean assessment to determine whether the LLM's answer appears within the list of all possible correct answers. If the LLM's answer matches any entry in the ground truth list, it is assigned a value of 1 (correct); if not, it receives a value of 0 (incorrect).
- **Q7:** This question is evaluated using the average R^2 score. It requires a series of numerical predictions from the LLM. By comparing these predictions to the ground truth values, we compute an R^2 score for each instance. The overall performance for all Q7-type questions is represented by the average of these R^2 scores.
- **Q8-Q12:** These questions involve binary classification tasks and are evaluated using the average bootstrap F1 score. Each task generates a single class prediction. We perform 1000 bootstrap sampling iterations with replacements on 50 versions of each question, generated from different instantiations of the question template. This process generates 1000 F1 scores. The average of these scores serves as a representation of performance for these binary classification tasks. We utilize F1 scores instead of accuracy to better address potential class imbalances in the data.
- **Q13:** This question is evaluated based on an exact match criterion. The performance metric is a boolean value indicating whether the LLM's answer matches exactly with the ground truth answer.

B Additional LIT Example

Figure 4 presents an illustrative case from Question 7 in which the model is asked to predict future patent application acceptance rates. For this forecasting task, the black-box solution uses the PandasInterpreter to compute grouped acceptance rates from previous months and employs an ARIMA model-based Forecaster to predict acceptance rates for the subsequent four months. In contrast, LIT generates predictions directly using the PandasInterpreter. While both methods yield suboptimal predictions (with an R^2 score close to zero) the LIT approach is significantly easier to debug. This is because the PandasInterpreter's code argument is transparent, making it evident that the logic of relying only on the first four months of historical data fails to address the forecasting question accurately. Conversely, debugging the ARIMA model-based Forecaster is more challenging, as the issues could stem from the data's lack of stationarity or inappropriate hyperparameter settings. By employing LIT to select the solution, the debugging process becomes more straightforward.

Question: First, group the patent applications by month from 2009 to 2012. For each month, calculate the percentage of applications that were accepted. Then, estimate the percentage of patents filed in the first 4 months of 2013 that will be accepted based on the monthly acceptance rates from the previous years. Return a list of acceptance percentages for each of the first 4 months of 2013, with each percentage expressed as a value between 0 and 100.

Without LIT

Solution

```
DBLoader(hupd, list(range(2009, 2013))),
PandasInterpreter(
    import pandas as pd
    df['year_month'] = df['filing_date'].dt.strftime('%Y-%m')
    grouped = df.groupby('year_month')
    acceptance_rates = df.groupby('year_month')['decision'].apply(
        lambda x: (x == 'ACCEPTED').sum() / len(x) * 100
    )
    acceptance_rate_list = acceptance_rates.tolist()
),
Forecaster(ARIMA, [74.02597402597402, 55.0561797752809, ... 68.42105263157895,
72.64957264957265], 4),
Finish({'forecast_predictions': [70.23170752205722, 69.06391351267285,
68.49988583683827, 68.22746859839476]}, forecast_predictions, list)
```

Cost = 3 + 2.236 + 8 + 0 = 13.236

With LIT

Solution 1

```
DBLoader(hupd, list(range(2009, 2013))),
PandasInterpreter(
    import pandas as pd
    df['year'] = df['filing_date'].dt.year
    df['month'] = df['filing_date'].dt.month
    grouped = df.groupby(['year', 'month'])
    acceptance_rates = grouped['patent_decision'].apply(
        lambda x: (x == 'ACCEPTED').sum() / len(x) * 100
    ).reset_index(name='acceptance_rate')
    early_acceptance_rates = acceptance_rates[
        acceptance_rates['month'] <= 4
    ].groupby('month')['acceptance_rate'].mean()
    predictions = list(early_acceptance_rates.values)
),
Finish({"result": [69.19, 62.26, 64.30, 66.57]}, result, list)
```

Cost = 3 + 3 + 0 = 6

Solution 2

```
DBLoader(hupd, list(range(2009, 2013))),
PandasInterpreter(
    import pandas as pd
    df['year_month'] = df['filing_date'].dt.strftime('%Y-%m')
    grouped = df.groupby('year_month')
    acceptance_rates =
    df.groupby('year_month')['decision'].apply(
        lambda x: (x == 'ACCEPTED').sum() / len(x) * 100
    )
    acceptance_rate_list = acceptance_rates.tolist()
),
Forecaster(linear_regression, [74.02597402597402, 55.0561797752809, ...
68.42105263157895, 72.64957264957265], 4),
Finish({'forecast_predictions': [74.7178511437216, 76.12632420865968,
77.53479727359776, 78.94327033853584]}, forecast_predictions, list)
```

Cost = 3 + 2.236 + 6 + 0 = 11.236

Solution 3

```
DBLoader(hupd, list(range(2009, 2013))),
PandasInterpreter(
    import pandas as pd
    df['year_month'] = df['filing_date'].dt.strftime('%Y-%m')
    grouped = df.groupby('year_month')
    acceptance_rates =
df.groupby('year_month')['decision'].apply(
        lambda x: (x == 'ACCEPTED').sum() / len(x) * 100
    )
    acceptance_rate_list = acceptance_rates.tolist()
),
Forecaster(ARIMA, [74.02597402597402, 55.0561797752809, ... 68.42105263157895,
72.64957264957265], 4),
Finish({'forecast_predictions': [70.23170752205722, 69.06391351267285,
68.49988583683827, 68.22746859839476]}, forecast_predictions, list)
```

Cost = 3 + 2.236 + 8 + 0 = 13.236

Figure 4: Solutions from an LLM to a question about future patent application acceptance, with and without LIT. When using LIT, the model generates multiple candidate solutions and selects the one with the best cost. As a result, the model with LIT chooses to use a simple average based on historical data rather than the ARIMA model used without LIT.

Act as a policy model to find the lowest total interpretability cost for solving a question with a given set of tools. Follow these steps: 1. Generate Solutions: List 2-4 sequences of tools that can solve the question. 2. Calculate and Compare Costs: Determine the total interpretability cost for each sequence. Prefer tools with lower costs. 3. Execute the Lowest Cost Solution.

Interpretability Costs:
Calculator: 2
DBLoader: 3
PandasInterpreter: √Lines of Code * max(Packages,1)
PythonInterpreter: Same as PandasInterpreter
Forecaster:
"linear_regression": 6
"ARIMA": 8
TextualClassifier:
"logistic_regression": 7
"cnn": 15
"bert-base-uncased": 20
LLMInferencer: 30
Finish: 0

Accuracy cannot be sacrificed for interpretability. Below are some examples that map the problem to the tools:

Question: What is the 20th Fibonacci number?
Solution:

```
PythonInterpreter(
    def solution(n):
        if n <= 0:
            return 0
        elif n == 1:
            return 1
        a, b = 0, 1
        for _ in range(2, n + 1
            a, b = b, a + b
        return b
    ans = solution(19)
),
Finish({'ans': 4181}, ans, integer)
```

Question: Which month had the highest number of patent applications in 2016?
Solution:

```
DBLoader(hupd, [2016]),
PandasInterpreter(
    import pandas as pd
    df['filing_month'] = df['filing_date'].apply(
        lambda x.month
    )
    month = df['filing_month'].mode()[0]
),
Finish({'month':12}, month, integer)
Finish({'ans': 'Advanced Techniques for 3D Scene Understanding and Adaptive Learning Models'}, ans, string)
```

Figure 5: The full prompt presented to models in the LIT framework (continued in Figure 6). Each LLM is presented with a set of costs, and instructed to form solutions that minimize these costs. A number of example solutions are provided.

C Full Prompt Details

Here we describe the full context given to each model in the LIT framework. Figures 5 and 6 present the complete context. When using LIT, we instruct each LLM to generate multiple candidate solutions for each problem and select the most reliable/inspectable one according to our specified cost functions. We then provide five examples of successful solutions.

Question: Determine if a NeurIPS paper, based on the following abstract, is assigned to Poster Session 2: 'We propose a Bayesian encoder for metric learning. Rather than relying on neural amortization as done in prior works, we learn a distribution over the network weights with the Laplace Approximation. We first prove that the contrastive loss is a negative log-likelihood on the spherical space. We propose three methods that ensure a positive definite covariance matrix. Lastly, we present a novel decomposition of the Generalized Gauss-Newton approximation. Empirically, we show that our Laplacian Metric Learner (LAM) yields well-calibrated uncertainties, reliably detects out-of-distribution examples, and has state-of-the-art predictive performance.' Return either '2' or 'not 2'.
Solution:

```
TextualClassifier(
    neurips,
    logistic_regression,
    Abstract,
    We propose a Bayesian encoder ... and has state-of-the-art predictive performance,
    Poster Session,
    2
),
Finish({'predictions': '2'}, predictions, string, ['2','not 2'])
```

Question: Using the patent applications from 2007 to 2009, predict the average length of claims for patent applications in 2010 and 2011.
Solution:

```
DBLoader(
    hupd,
    list(range(2007,2010))
),
PandasInterpreter(
    import pandas as pd
    df['year'] = df['filing_date'].dt.year
    df['len_claims'] = df['claims'].apply(len
    average_claims_per_year = df.groupby('year')['len_claims'].mean())
),
Forecaster(
    linear_regression,
    previous_data,
    2
),
Finish({'forecast_predictions': [6020.225608051151, 5998.883671776641]}, forecast_predictions, list)
```

Question: Identify a common theme that links the NeurIPS papers titled '4D Panoptic Scene Graph Generation,' 'VoxDet: Voxel Learning for Novel Instance Detection,' and 'L2T-DLN: Learning to Teach with Dynamic Loss Network.'
Solution:

```
LLMInferencer(),
Finish({'ans': 'Advanced Techniques for 3D Scene Understanding and Adaptive Learning Models'}, ans, string)
```

Figure 6: The full prompt presented to models in the LIT framework (continued from Figure 5). Each LLM is presented with a set of costs, and instructed to form solutions that minimize these costs. A number of example solutions are provided.

Tool	P	D	C	Cost=P+D+C
Calculator	0	1	1	2
DBLoader	0	2	1	3
PythonInterpreter	0	$\sqrt{\text{lines}} \times \max(\text{packages}, 1) \times 0.5$	$\sqrt{\text{lines}} \times \max(\text{packages}, 1) \times 0.5$	$\sqrt{\text{lines}} \times \max(\text{packages}, 1)$
PandasInterpreter	0	$\sqrt{\text{lines}} \times \max(\text{packages}, 1) \times 0.5$	$\sqrt{\text{lines}} \times \max(\text{packages}, 1) \times 0.5$	$\sqrt{\text{lines}} \times \max(\text{packages}, 1)$
Forecaster (linear regression)	3	2	1	6
Forecaster (ARIMA)	4	3	1	8
TextualClassifier (logistic regression)	3	2	2	7
TextualClassifier (cnn)	2	7	6	15
TextualClassifier (BERT)	2	10	8	20
LLMInferencer	1	15	14	30
Finish	0	0	0	0

Table 3: Components of the cost in our experiments. Users of the LIT framework can choose their own tool costs based on their preferences. Cost components P, D, C refer to Robust Performance Across Inputs, Ease of Debugging, and Complexity of Arguments. In our experiments, we chose simple tools, such as the calculator, to have low P cost. PythonInterpreter and PandasInterpreter increase in cost when their results contain more lines or more packages.

D Detailed Description of Question Bank

Below are the 13 predefined question templates, each of which generates 100 variations by substituting the values inside the curly braces {}. Seven of these questions are related to the HUPD dataset, while six are related to the NeurIPS 2023 Papers Dataset, as noted in the brackets []. The questions are categorized by difficulty as follows: Q1-Q6 are "easy," Q7-Q10 are "medium," and Q11-Q13 are "hard."

- Q1. [HUPD] What was the average time between the filing and issuance of patents from {start_year} to {end_year}? Return an integer representing the number of days by truncating the decimal part.
- Q2. [HUPD] What were the top {#} {IPCR/CPC categories} with the highest number of accepted patents in {year}? Return them as a list of {IPCR/CPC categories}.
- Q3. [HUPD] How does the number of patent applications filed in {year1} compare proportionally to those filed in the {year2}? Return a number between 0 and 1. Please note that each row represents a patent application, and not all patent applications are assigned a patent number.
- Q4. [HUPD] What is the title of the patent filed between {start_year} and {end_year} that took the longest number of days between the filing date and the publication date?
- Q5. [NeurIPS] Who were the top {#} authors with the most publications {where the titles contain 'Large Language Models'} at NeurIPS?
- Q6. [NeurIPS] What proportion of NeurIPS papers have {compare} {n} authors? In the authors column of the database, each entry is a list, not a single string. Return a value between 0 and 1.
- Q7. [HUPD] First, group the patent applications by month from {start_year} to 2012. For each month, calculate the percentage of applications that were accepted. Then, estimate the percentage of patents filed in the first {n} months of 2013 that will be accepted based on the monthly acceptance rates from the previous years. Return a list of acceptance percentages for each of the first {n} months of 2013, with each percentage expressed as a value between 0 and 100.
- Q8. [HUPD] For a patent application, which is not present in the database, with an abstract {abstract_content}, predict whether it will get accepted. Return either 'ACCEPTED' or 'not ACCEPTED.'
- Q9. [NeurIPS] For a NeurIPS paper, which is not present in the database, with title {title_content}, predict whether it belongs to {topic}? Return either '{topic}' or 'not {topic}'.
- Q10. [NeurIPS] For a NeurIPS paper, which is not present in the database, with abstract {abstract_content}, predict whether it will be accepted as an oral presentation? Return either 'oral' or 'not oral'.

- Q11. [HUPD] Predict if the following two patents, which are not present in the database, belong to the same CPC category: {title1}, {title2}? Return 'Yes' or 'No'.
- Q12. [NeurIPS] Predict if this abstract-title pair, which is not present in the database, is from the same NeurIPS paper: Abstract: {abstract}. Title: {title}. Return 'Yes' or 'No'.
- Q13. [NeurIPS] Predict the best fit topic for the title of a NeurIPS paper, which is not present in the database: {title}. Options: {topic1}, {topic2}, {topic3}.

E Details of Model-Based Tools

- **Forecaster:**
  - **linear_regression:** Utilizes the default settings of scikit-learn's linear regression model.
  - **ARIMA:** Configured as a first-order autoregressive model (p=1), first-order differencing (d=1), and first-order moving average model (q=1).
- **TextualClassifier:**
  - **bert-base-uncased, cnn, logistic_regression:** Uses the default configurations of the respective models.
  - **hupd:** Further trained on patents filed between 2004 and 2012. Related questions are based on the held-out set of patents filed between 2013 and 2018.
  - **neurips:** Further trained on the first 3,000 papers from the NeurIPS 2023 Papers Dataset. Related questions are based on the held-out set of the last 590 papers in the dataset.