I Need Help! Evaluating LLM's Ability to Ask for Users' Support: A Case Study on Text-to-SQL Generation

Cheng-Kuang Wu[1,2*], **Zhi Rui Tam**[1*], **Chao-Chung Wu**[1], **Chieh-Yen Lin**[1], **Hung-yi Lee**[2†], **Yun-Nung Chen**[2†]
[1]Appier AI Research
[2]National Taiwan University

Abstract

This study explores the proactive ability of LLMs to seek user support. We propose metrics to evaluate the trade-off between performance improvements and user burden, and investigate whether LLMs can determine when to request help under varying information availability. Our experiments show that without external feedback, many LLMs struggle to recognize their need for user support. The findings highlight the importance of external signals and provide insights for future research on improving support-seeking strategies. Source code: https://github.com/appier-research/i-need-help.

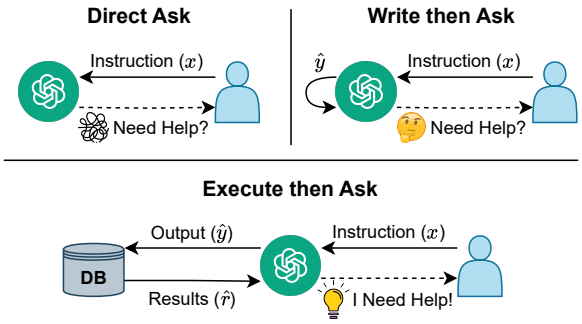

Figure 1: Overview of our experiments on text-to-SQL. LLMs struggle to determine when they need help based solely on the instruction (x) or their output ($\hat{y}$). They require external feedback, such as the execution results ($\hat{r}$) from the database, to outperform random baselines.

1 Introduction

The impressive instruction-following (Wei et al., 2021) abilities of large language models (LLMs) have enabled their out-of-the-box usage to solve problems. However, these models generate hallucinated content (Rawte et al., 2023) or incorrect predictions in their efforts to fulfill user instructions, which undermines their reliability.

When LLMs generate incorrect outputs for a given instruction, the issue can be examined from multiple perspectives. One is that the model simply lacks the *competence* to satisfy the instruction, suggesting a straightforward solution: enhancing the model's capabilities, which is the focus of most previous research. Another is that the model could actually solve the task with additional *support*. For instance, Pourreza and Rafiei (2023) found that models often fail due to underspecified natural language queries. Similarly, Li et al. (2024) showed that while GPT-4 struggles initially, its performance can improve by up to 20.01% with human-annotated external knowledge. In such cases, models should proactively seek help rather than attempting to satisfy instructions with insufficient information.

Motivated by these considerations, we aim to investigate whether LLMs can identify when to ask for user support. Since providing such support requires additional effort from users, there is an inherent trade-off between "LLM performance improvement from user support" and "user burden". Therefore, we seek to answer the following research questions: ***RQ1:*** *How can we design evaluation metrics to quantify this trade-off?* ***RQ2:*** *How effectively do LLMs manage this trade-off, and what strategies are effective in improving it?*

In this work, we focus on the text-to-SQL task as a case study to empirically investigate the aforementioned research questions. We chose the text-to-SQL task for several reasons: (1) Its promising applicability, empowering lay users to retrieve data with natural language queries. (2) The inherent ambiguity in some natural language queries, leading to uncertainty in the generation of SQL code (Pourreza and Rafiei, 2023), making it suitable for scenarios where additional user support is beneficial. (3) There exists a large-scale BIRD dataset (Li et al., 2024) with human-annotated external knowledge, providing a valuable source of user support for our empirical investigation.

[*]Equal contribution
[†]Equal advisorship

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 2191–2199
November 12-16, 2024

Our contributions can be summarized as follows:

1. We propose metrics for evaluating the trade-off between performance improvement from user support and the associated user burden.
2. We conduct experiments using various methods to balance this trade-off, providing insights into LLMs' capabilities in seeking user support and identifying effective strategies for enhancing their performance.

2 Formulation for Seeking Support

2.1 General Setup

Consider an LLM f parameterized by θ, along with a prompt template $p(\cdot)$. Given a natural language instruction x, we use z to represent *support*, which should enhance the LLM's ability to fulfill x. Formally, $\hat{y}_z = f(p(x, z) \mid \theta)$ is more likely to satisfy x compared to $\hat{y} = f(p(x) \mid \theta)$. We denote the "ask for support" signal emitted by the LLM as $\hat{a}$, defined as a confidence score in the range $[0, 1]$, where 1 indicates an absolute need for support. A threshold τ is then used to determine whether to request z. In practice, $\hat{a}$ could also be a natural language request specifying the *type of support* needed by the LLM, which we leave for future work.

2.2 Evaluation

To measure the trade-off between *performance improvement from user support* and *user burden*, we need 2-dimensional evaluation. One dimension is the user burden (B), which we define as the proportion of instances where the LLM ask for support:

$$B = \frac{N_{\text{ask}}}{N}$$

where N_{ask} is the number of instances where the LLM asks for support, and N denotes the total number of instances in the test set. The other dimension is the performance improvement (Δ, Delta):

$$\Delta = \frac{1}{N} \sum_{i=1}^{N_{\text{ask}}} (h(y_i, \hat{y}_{i,z}) - h(y_i, \hat{y}_i))$$

where $h(\cdot)$ is the evaluation function of a given task, which takes ground truth y_i and model output $\hat{y}_i$ as arguments ($\hat{y}_{i,z}$ is an output with the help of z). Inspired by the idea behind the ROC curve (Majnik and Bosnić, 2013), we illustrate this trade-off with a graph, where the performance curve is plotted by adjusting the threshold τ from high to low along the x-axis. We refer to this curve as Delta-Burden Curve (DBC) (see the leftmost subplot of Figure 2).

2.3 Methods for Seeking Support

We design a prompt template $p_{\text{ask}}(\cdot)$ to enable LLMs to request support by $\hat{a} = s(f(p_{\text{ask}}(w) \mid \theta))$. Here, w represents the textual information that the LLM f uses to determine whether it needs to seek support, and s is the scoring function that converts the probability distribution of output tokens into a confidence score $\hat{a} \in [0, 1]$. We propose methods with varying compositions of w to explore the information LLMs require to achieve better trade-off under DBC. Note that p_{ask} remains the same across all methods to minimize prompt engineering. An overview of these methods is shown in Figure 1.

Direct Ask (DA): $w = (db, x)$, composed of database schema db and user data requirement x.

Write then Ask (WA): $w = (db, x, \hat{y})$, where the LLM generates the SQL code $\hat{y} = f(p(db, x) \mid \theta)$ first and then use this self-generated output as the additional information in w.

Execute then Ask (EA): $w = (db, x, \hat{y}, \hat{r})$, where the execution results $\hat{r}$ is returned by the database by executing LLM-generated SQL $\hat{y}$.

3 Experiments

3.1 Dataset

We use BIRD (Li et al., 2024), which includes human-annotated external knowledge that serves as z. For example, z might be domain-specific knowledge, such as how to calculate financial indicators from database values. The instruction x represents the users' data requirements, paired with the ground truth SQL y. It uses **Execution Accuracy (EX)** as the evaluation metric, where $h(y_i, \hat{y}_i)$ is defined as $\mathbb{1}(r_i = \hat{r}_i)$. Here, r_i is the SQL execution result of y_i, and $\hat{r}_i$ is the execution result of $\hat{y}_i$. Simply put, EX is the proportion of testing instances where r_i and $\hat{r}_i$ are identical.

3.2 Implementation

For open-weight LLMs, we use *WizardCoder-34B* (Luo et al., 2023), *Llama-3-70b-chat*, *DeepSeek-Coder-33B* (Guo et al., 2024), and *Mixtral-8x22B* (Jiang et al., 2024) for diversity of different LLM families. For closed-source LLMs, we use *gpt-3.5-turbo-0125*, *gpt-4-turbo-2024-04-09*, and *gpt-4o-2024-05-13* (OpenAI, 2023). The prompt $p_{\text{ask}}(w)$ (included in Appendix A) instructs the model to output a single token `Yes/No` to indicate whether it needs support. We define the scoring function s as the softmax of `Yes` over log probabilities of `Yes` and `No` to derive $\hat{a} \in [0, 1]$.

Methods/LLMs	Wizard	Llama3	DPSeek	GPT-3.5	Mixtral	GPT-4t	GPT-4o
Random Baseline	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000
Direct Ask	0.4915	0.4834	0.4976	0.4390	0.5301	0.5758	0.5479
Write then Ask	0.4759	0.4497	0.4857	0.4735	0.5677	0.5807	0.5740
Execute then Ask	**0.5096**	**0.4987**	**0.5848**	**0.6313**	**0.6242**	**0.6641**	**0.5989**

Table 1: Area Under Delta-Burden Curve (AUDBC) across different methods and LLMs. Text in **bold** denotes the method with the best performance, while underlined text means better than random (uniform sampling of $\hat{a} \in [0, 1]$).

Support/LLMs	Wizard	Llama3	DPSeek	GPT-3.5	Mixtral	GPT-4t	GPT-4o
w/o user support	0.1721	0.1767	0.2360	0.3064	0.2419	0.3142	0.3096
w/ full user support	0.2764	0.3475	0.4185	0.4668	0.4126	0.4889	0.5117

Table 2: Execution accuracy (EX) of different support levels. Full user support means $B = 1$ (see Section 2.2).

4 Main Results

Using the formulation in Section 2.2, we quantify the performance of different methods with the Area Under Delta-Burden Curve (AUDBC) in Table 1. Visualized DBCs are available in the leftmost subplots in Figure 2. Note that AUDBC should only be compared between methods under the same LLM, as it is normalized to the range of $[0, 1]$ by dividing the area under the curve by the maximum square area, which depends on the scale of ΔEX and differs across LLMs, as shown in Table 2.

There are three major findings: (1) Execution then Ask consistently improves the performance-burden trade-off for LLMs, although *Llama-3-70b-chat* fails to outperform the random baseline. (2) The leftmost four LLMs in Table 1 do not surpass the random baseline without the assistance of $\hat{r}$, indicating that many current LLMs still struggle to determine the need for support based on x and $\hat{y}$ alone. (3) Despite this, the rightmost three LLMs outperform the random baseline with the Write then Ask $(x, \hat{y})$ or even Direct Ask (x) methods. Nevertheless, the inclusion of $\hat{r}$ remains beneficial for further enhancing the trade-off between performance improvement and user burden. Practical implications of the third point include the potential for cost savings by trading off the execution of $\hat{y}$ to obtain $\hat{r}$ in certain resource-constrained scenarios.

5 Discussion

5.1 Analysis on the Delta-Burden Curves

The Delta-Burden Curves (DBCs) plotted in Figure 2 quantify the following practical question: *Under the same user burden, which method can achieve more performance boost?* To further analyze how this performance boost is achieved, we decompose the concept into two abilities:

1. The ability to ask for support when the LLM cannot satisfy the instruction originally.
2. The ability to utilize support effectively to *flip* the incorrect output to the correct output.

1. For the first ability, we introduce the following metrics inspired by the precision-recall trade-off:
Precision of Asking for Support (P_{ask}) When the LLM asks for support, it should be the case that the LLM cannot satisfy the instruction originally, or it would cause unnecessary user burden:

$$P_{\text{ask}} = \frac{\#(\text{AskforSupport} \ \& \ \text{OriginallyWrong})}{\#\text{AskforSupport}}$$

Recall of Asking for Support (R_{ask}) When the LLM is not able to satisfy the instruction originally, it should identify this need and ask for support:

$$R_{\text{ask}} = \frac{\#(\text{AskforSupport} \ \& \ \text{OriginallyWrong})}{\#\text{OriginallyWrong}}$$

PR Curve of Asking for Support Similar to how DBC is plotted, one can also adjust the threshold $\tau \in [0, 1]$ from high to low along the x-axis to plot the Precision-Recall Curve of Asking for Support.
2. For the second ability, we introduce *Flip Rate*:
Flip Rate: This metric is calculated as the proportion of instances where the LLM's initially incorrect answers were corrected after receiving support, divided by the total number of instances where support was requested. Formally, it is defined as:

$$FR = \frac{1}{N_{ask}} \sum_{i=1}^{N_{\text{ask}}} (h(y_i, \hat{y}_{i,z}) - h(y_i, \hat{y}_i))$$

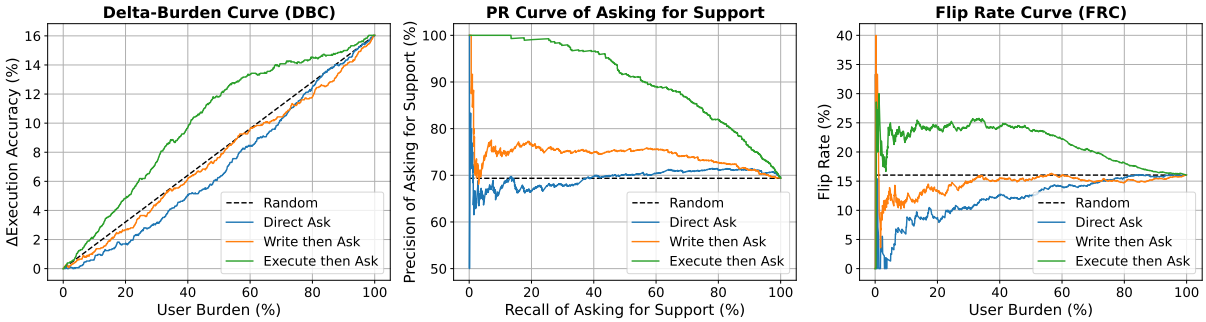

Figure 2: Performance curves of *gpt-3.5-turbo-0125*. Curves of other LLMs are shown in Appendix B.

Methods/LLMs	Wizard	Llama3	DPSeek	GPT-3.5	Mixtral	GPT-4t	GPT-4o	Gemini	Claude
Random Baseline	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000
EA (real logprobs)	**0.5096**	0.4987	**0.5848**	**0.6313**	**0.6242**	**0.6641**	**0.5989**	-	-
EA (verbalized)	0.5011	**0.5333**	0.4964	0.5945	0.6226	0.4850	0.5152	**0.5624**	**0.6174**

Table 3: Area Under Delta-Burden Curve (AUDBC) with the verbalized token log probabilities approach. Text in **bold** denotes the method with the best performance, while underlined text means better than random.

Different from Δ defined in Section 2.2, this metric emphasizes the *efficiency* of leveraging support instead of the total improvement on the test set. Like DBC, one may adjust the threshold τ to plot the Flip Rate Curve (FRC). With the definition of these two abilities, we plot the DBC, PR Curve, and FRC on Figure 2. Although the Write then Ask method shows near-random performance in DBC, the PR Curve indicates it achieves better-than-random performance in identifying when support is needed. However, its lower Flip Rate suggests it is less efficient in utilizing the support to correct mistakes. These two abilities, represented by the PR Curve and FRC, respectively, balance each other out, resulting in near-random performance on the DBC. This finding shows that the ability to identify the need for support and the ability to utilize that support are distinct. In future work, it is worth exploring how to further enhance each of these abilities.

5.2 LLMs without Access to Log Probabilities

Given that not all LLMs provide access to token log probabilities, we discuss how our method can be adapted for these "black-box" models. We modify the prompt template p_{ask} to p_{verb}, which instructs the LLM to output the *verbalized* confidence score $\hat{a}$ directly by specifying the range and meaning of $\hat{a} \in [0, 1]$ in p_{verb} (attached in Appendix A.2). In addition to the seven LLMs mentioned in Section 3.2, we also include two black-box models: *gemini-1.0-pro-001* and *claude-3-haiku-20240307*. The results, shown in Table 3, indicate that using verbalized confidence scores generally degrades performance for most LLMs. However, it remains a promising alternative for black-box LLMs such as Gemini and Claude to surpass the random baseline.

6 Related Work

The ability of LLMs to identify the need for support relies on their *well-calibratedness* (Kadavath et al., 2022), which refers to their capacity to recognize uncertainty. Previous studies focus on enhancing the calibration of predictions (Xiao et al., 2022; Kuhn et al., 2023), or using verbalized token probabilities to achieve better calibration (Tian et al., 2023). Our work extends this line of research by exploring how LLMs can effectively seek user support by leveraging their well-calibrated property. The major distinction between this and existing calibration studies lies in extending the focus from *identifying* the uncertainty to *utilizing* support.

7 Conclusion

We propose a framework for LLMs to seek support, and evaluate methods on Text-to-SQL generation. Our findings suggest the importance of external signals, such as SQL execution results, in helping LLMs better manage performance-burden trade-off. We further decompose DBC into the ability of *identify* the need for support and the ability to *utilize* the support. Future works may explore a broader range of tasks or develop methods to improve both the identification and utilization of support.

8 Limitations

8.1 Task Coverage

The scope of our experiments is limited to the Text-to-SQL task. While this task provides a useful case study for evaluating LLMs' ability to seek and utilize support, it does not encompass the full range of potential applications for LLMs. Future work should extend the evaluation to a broader set of tasks to ensure the generalizability of our findings.

8.2 Types of Support

In this study, we primarily focus on a single type of support: human-annotated external knowledge. However, there are many other types of support that LLMs might require. Future works could explore how LLMs can request and utilize these various forms of support to enhance their performance.

8.3 Dependence on External Feedback

Our findings indicate that LLMs significantly benefit from external signals, such as SQL execution results. However, this reliance on external feedback may not always be feasible in practical applications, where immediate execution or access to external data might be limited. Developing methods that enable LLMs to better manage without such feedback remains an important area for future exploration.

References

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming - the rise of code intelligence. *ArXiv*, abs/2401.14196.

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, Gianna Lengyel, Guillaume Bour, Guillaume Lample, L'elio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts. *ArXiv*, abs/2401.04088.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, et al. 2022. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2023. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. *arXiv preprint arXiv:2302.09664*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. *ArXiv*, abs/2306.08568.

Matjaž Majnik and Zoran Bosnić. 2013. Roc analysis of classifiers in machine learning: A survey. *Intelligent data analysis*, 17(3):531–558.

OpenAI. 2023. Gpt-4 technical report.

Mohammadreza Pourreza and Davood Rafiei. 2023. Evaluating cross-domain text-to-sql models and benchmarks. *arXiv preprint arXiv:2310.18538*.

Vipula Rawte, Amit Sheth, and Amitava Das. 2023. A survey of hallucination in large foundation models. *arXiv preprint arXiv:2309.05922*.

Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher D Manning. 2023. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5433–5442.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Yuxin Xiao, Paul Pu Liang, Umang Bhatt, Willie Neiswanger, Ruslan Salakhutdinov, and Louis-Philippe Morency. 2022. Uncertainty quantification with pre-trained language models: A large-scale empirical analysis. *arXiv preprint arXiv:2210.04714*.

A Prompt Templates

We include the prompt templates used in this work.

A.1 Prompt for Seeking Support

The prompt template $p_{\text{ask}}(w)$ used to instruct LLMs for seeking support is as follows:

> You are currently doing the text-to-SQL task. Based on the information provided ({items}), you have to determine whether additional hints are required for you to generate the SQL correctly to answer the user's question. You should only ask for additional hints when you actually need them, since you will also be evaluated based on the number of times you ask for hints, which would be provided by the user.
>
> information provided (enclosed by triple backticks):
> ```
> {information}
> ```
>
> Answer a single word Yes if you need hints (since the information provided is not enough to generate SQL correctly). Answer a single word No if hints are not required (since you are already confident to generate SQL).
> Do you need additional hints? Answer (Yes / No):

In this template, the actual contents of {items} and {information} depend on the method used. The contents are summarized in Table 4. For example, $w = (db, x, \hat{y}, \hat{r})$ in Execute then Ask (EA), so {items} will be filled with the four item names and {information} will be replaced by actual information of the four items. Similarly for Write then Ask ($w = (db, x, \hat{y})$) and Direct Ask ($w = (db, x)$).

Item	Item Name	Information
db	Database schema	{db_schema}
x	User's question	{question}
$\hat{y}$	Generated SQL	{gen_sql}
$\hat{r}$	SQL execution results	{exe_results}

Table 4: Contents in the prompt $p_{\text{ask}}(w)$, where {items} will be filled with words in the "Item Name" column, while {information} is replaced with actual information of text in {red}.

A.2 Prompt for Seeking Support (Verbalized)

The prompt template for generating verbalized probabilities in LLMs without access to token log probabilities (e.g., Gemini and Claude families):

> You are currently doing the text-to-SQL task. Based on the information provided ({items}), you have to determine whether additional hints are required for you to generate the SQL correctly to answer the user's question. You should only ask for additional hints when you actually need them, since you will also be evaluated based on the number of times you ask for hints, which would be provided by the user.
>
> information provided (enclosed by triple backticks):
> ```
> {information}
> ```
>
> Do you need additional hints? Provide the precise probability that you need hints (closer to 0 means you don't need hints, closer to 1 means you need hints).
> Give ONLY the precise probability to five decimal places (format: 0.abcde, where abcde can be different digits), no other words or explanations are needed.

The prompt template is similar to the original template shown in A.1, except that the last few sentences are modified.

A.3 Prompt for Generating SQL Code

The prompt template $p(\cdot)$ for converting user data requirement x into SQL code is as follows:

> {db_schema}
>
> – Using valid SQLite, answer the following questions for the tables provided above.
> – Question: {question}
> Now, generate the correct SQL code directly in the format of "'sql\n<your_SQL_code>\n"':

If user support z is provided (i.e., when LLMs ask for support), the prompt template is slightly modified as follows:

> {db_schema}
>
> – External Knowledge: {support}
> – Using valid SQLite, answer the following questions for the tables provided above. You can use the provided External Knowledge to help you generate valid and correct SQLite.
> – Question: {question}
> Now, generate the correct SQL code directly in the format of "'sql\n<your_SQL_code>\n"':

In these two templates, {db_schema} is db, {question} is user data requirement x, and {support} is user support z, which is human-annotated external knowledge in BIRD (Li et al., 2024).

B Performance Curves

We present visualizations of all performance curves in Table 3, 4, 5, 6, 7, 8, and 9.

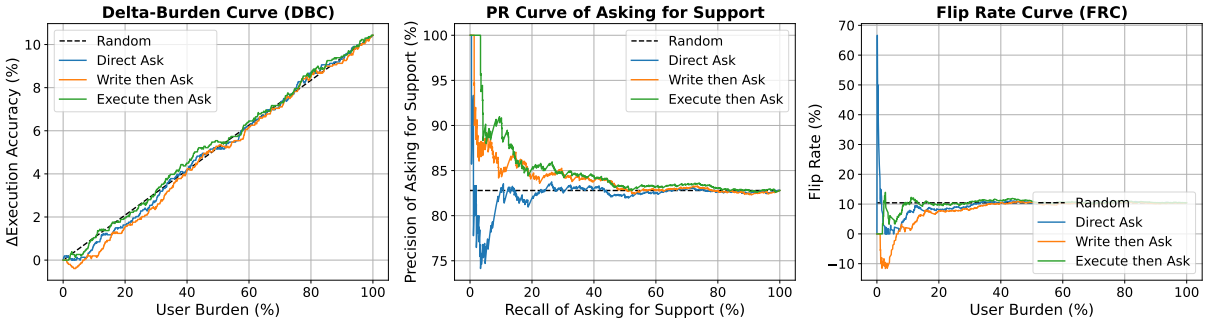

Figure 3: Performance curves of *WizardCoder-Python-34B-V1.0*.

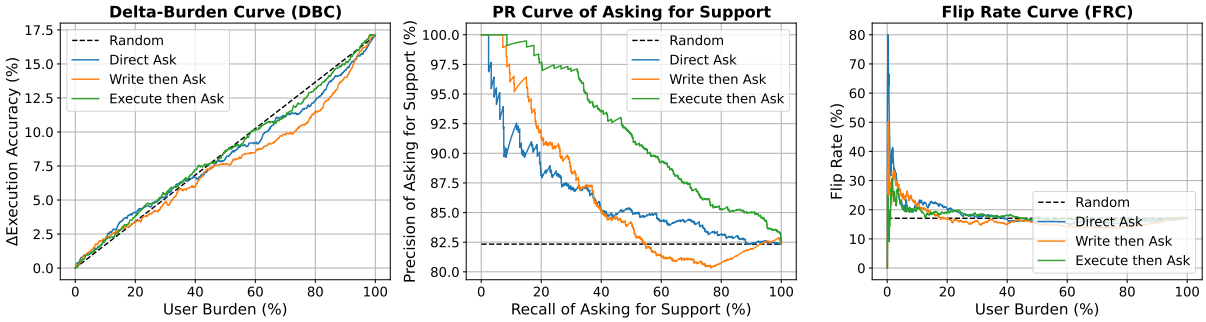

Figure 4: Performance curves of *Llama-3-70b-chat-hf*.

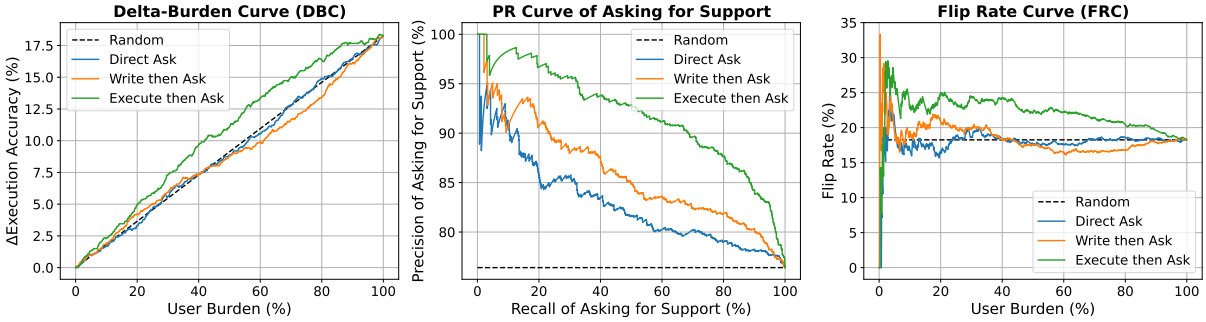

Figure 5: Performance curves of *deepseek-coder-33b-instruct*.

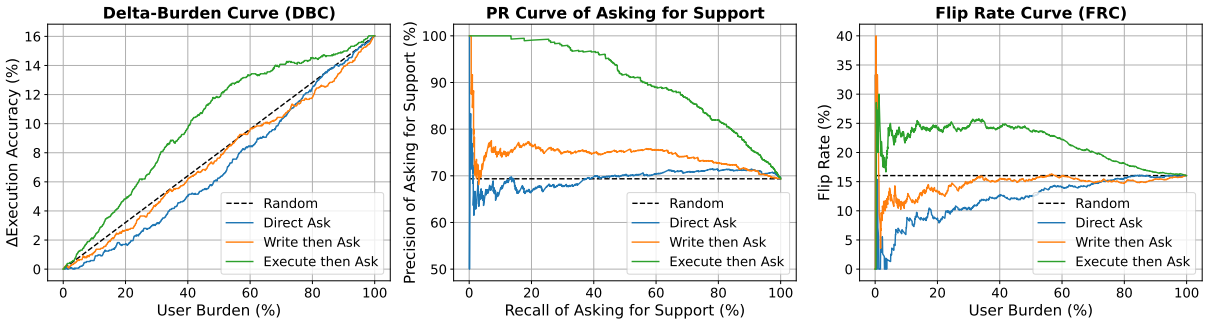

Figure 6: Performance curves of *gpt-3.5-turbo-0125*.

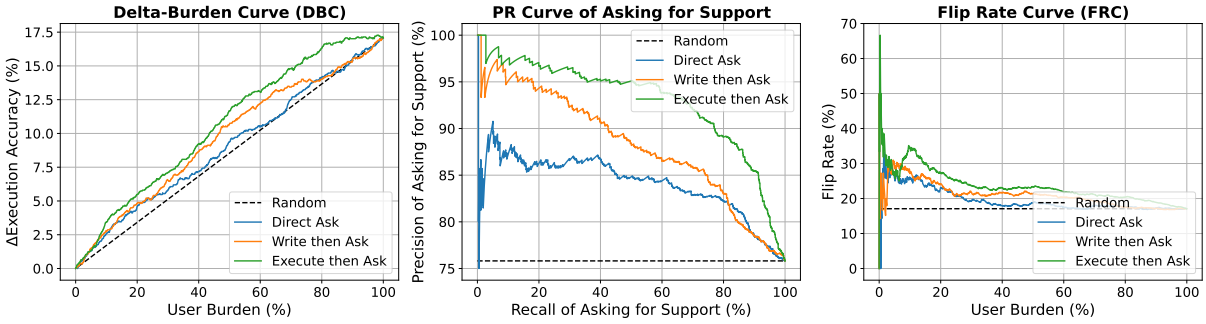

Figure 7: Performance curves of *Mixtral-8x22B-Instruct-v0.1*.

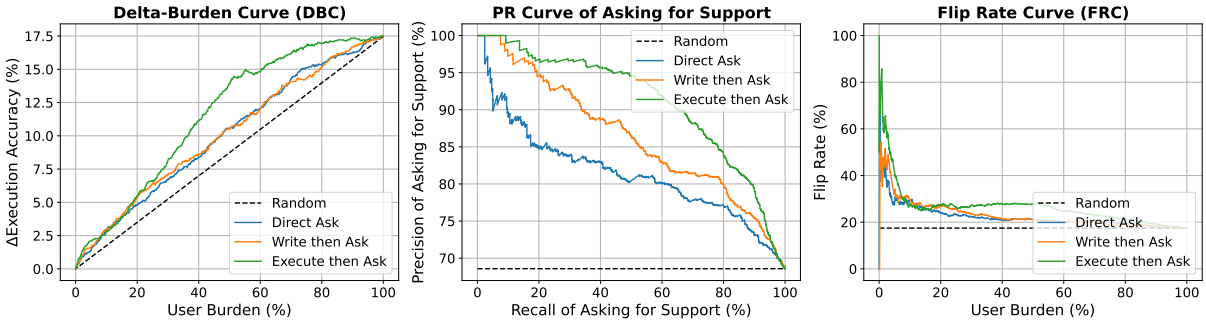

Figure 8: Performance curves of *gpt-4-turbo-2024-04-09*.

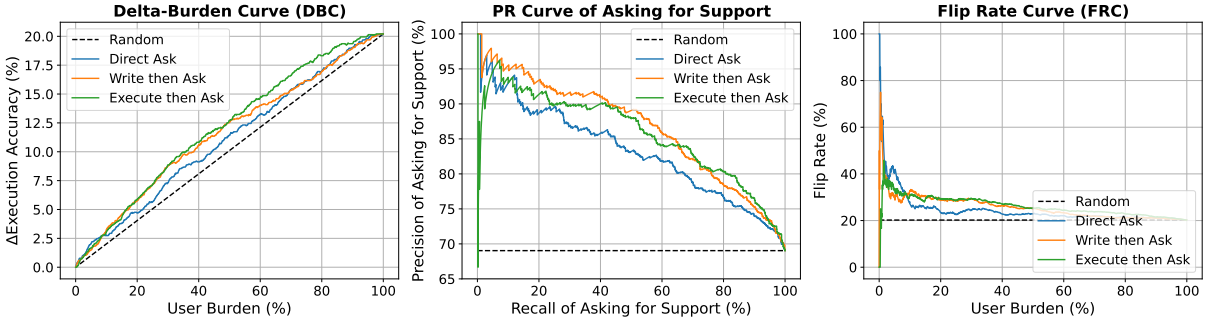

Figure 9: Performance curves of *gpt-4o-2024-05-13*.