CMAT: A Multi-Agent Collaboration Tuning Framework for Enhancing Small Language Models

Anonymous ACL submission

Abstract

Open large language models (LLMs) have significantly advanced the field of natural language processing, showcasing impressive performance across various tasks. Despite the significant advancements in LLMs, their effective operation still relies heavily on human input to accurately guide the dialogue flow, with agent tuning being a crucial optimization technique that involves human adjustments to the model for better response to such guidance. Addressing this dependency, our work introduces the TinyAgent model, trained on a meticulously curated high-quality dataset. We also present the Collaborative Multi-Agent Tuning (CMAT) framework, an innovative system designed to augment language agent capabilities through adaptive weight updates based on environmental feedback. This framework fosters collaborative learning and real-time adaptation among multiple intelligent agents, enhancing their context-awareness and long-term memory. In this research, we propose a new communication agent framework that integrates multi-agent systems with environmental feedback mechanisms, offering a scalable method to explore cooperative behaviors. Notably, our TinyAgent-7B model exhibits performance on par with GPT-3.5, despite having fewer parameters, signifying a substantial improvement in the efficiency and effectiveness of LLMs.

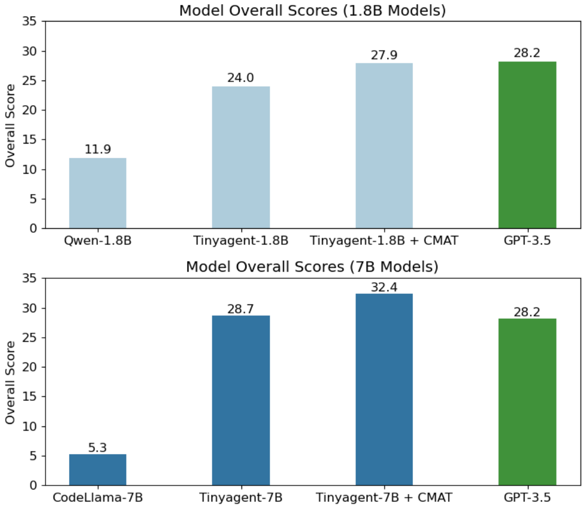


Figure 1: TinyAgent demonstrates outstanding performance, comparable to that of GPT-3.5. TinyAgent is a series of models fine-tuned based on Qwen (Bai et al., 2023) and Codellama (Roziere et al., 2023).

1 Introduction

In the rapid development of the field of artificial intelligence, large language models (LLMs) such as BERT and GPT-4 (OpenAI, 2023) have become important cornerstones of natural language processing (NLP). These models utilize the Transformer architecture and effectively capture long-distance dependencies through multi-head self-attention mechanisms, demonstrating strong capabilities across various NLP tasks. With technological advancements, the performance and application scope of LLMs continue to expand, promising significant improvements in computational efficiency and functionality, including anticipated advanced features such as self-improvement, self-checking, and sparse expert models (Liu et al., 2023).

However, it is noteworthy that the success of these models largely depends on human input to guide the correct dialogue. This dependency requires users to provide relevant and precise prompts based on their intentions and the feedback from the chat agent, raising a critical question: ***Can we replace human intervention with autonomous communication agents capable of steering conversations towards task completion with minimal human supervision?***

Our research is driven by the need to overcome the significant challenges faced by LLMs in real-world deployments, particularly the high computational resource requirements, data biases, and lack of robustness. These issues limit their applicability in resource-constrained environments and highlight the urgency of enhancing model efficiency and adaptability (Abid et al., 2021; Du et al., 2022). As demonstrated by Figure 1, we aim to address these limitations by optimizing models and training methods to enable smaller models to match the performance levels of larger models. Additionally, recognizing the potential of MAS to improve processing efficiency and system adaptability through agent cooperation, we seek to develop a collaborative agent framework. This framework aims to facilitate effective cooperation among agents, thereby overcoming the performance gap and propelling further research and innovation in the field of LLMs (Ferry et al., 2018; Talwar et al., 2005). In our experiments, we evaluated the capabilities of large models with and without the use of prompts and observed that low-quality prompts can significantly degrade model performance. Consequently, we propose the Collaborative Multi-Agent Tuning (CMAT) framework.

The CMAT framework introduces a structured environment where individual agents, each with specialized roles and capabilities, work together to process information, make decisions, and solve complex tasks (Hernández-Orallo et al., 2017). By sharing insights and learning from interactions within this multi-agent ecosystem, the framework allows for a more scalable and flexible approach to training LLMs (Lewis et al., 2017). This collaborative effort not only helps in bridging the gap in performance between smaller and larger models but also fosters a more resilient system capable of adapting to new challenges without extensive human intervention (Kaplan et al., 2020). Through CMAT, we aim to push the boundaries of what is possible with LLMs, making them more accessible and effective for a wider range of applications (Rajpurkar et al., 2018).

The main contributions of our work are as follows:

- We propose the CMAT framework which represents an innovative approach that allows for dynamic and real-time memory updates within multi-agent systems.
- We design a novel role-playing mechanism for precise task allocation and enhanced agent communication, significantly boosting overall performance and cooperation.
- We evaluated the fine-tuned TinyAgent models across multiple agent tasks, finding that in certain scenarios, their performance rivals that of advanced LLMs like GPT-4 and agentlm (Zeng et al., 2023), demonstrating the potential efficiency and capabilities of compact models.

2 Related Work

2.1 LLMs Applications in a Multi-Agent Framework

We explore the applications of LLMs within multi-agent systems, highlighting their role versatility as users, assistants, and checkers, and their capability to offer bespoke support and solutions across such environments (de Zarzà et al., 2023; Talebirad and Nadiri, 2023). LLMs showcase remarkable adaptability to tasks through methods like supervised fine-tuning and real-time feedback learning, notably in tasks that require a sophisticated understanding and execution related to operating systems or databases (Christianos et al., 2023; Li et al., 2023). Furthermore, LLMs are adept at enhancing communication and collaboration among agents, a critical component for addressing complex issues that necessitate multi-role coordination (Zhao et al., 2021). Nevertheless, LLMs encounter specific challenges within multi-agent frameworks, especially in situations that demand a nuanced contextual comprehension and sustained memory retention, as well as adapting to fast-evolving environments and unforeseeable tasks (Diallo et al., 2020). Issues such as data bias, security concerns, and the intricacies of crafting effective protocols for multi-agent cooperation stand as significant hurdles in this domain (Zhang et al., 2017; García et al., 2015). Thus, by summarizing LLMs' roles in multi-agent frameworks, we underscore the critical need for continued innovation and research exploration, aimed at overcoming these technological hurdles and leveraging the full potential of LLMs in complex systems (Lu and Zhang, 2020).

To enhance the adaptability and collaborative capabilities of LLMs in multi-agent systems, we've implemented memory modes, including long-term

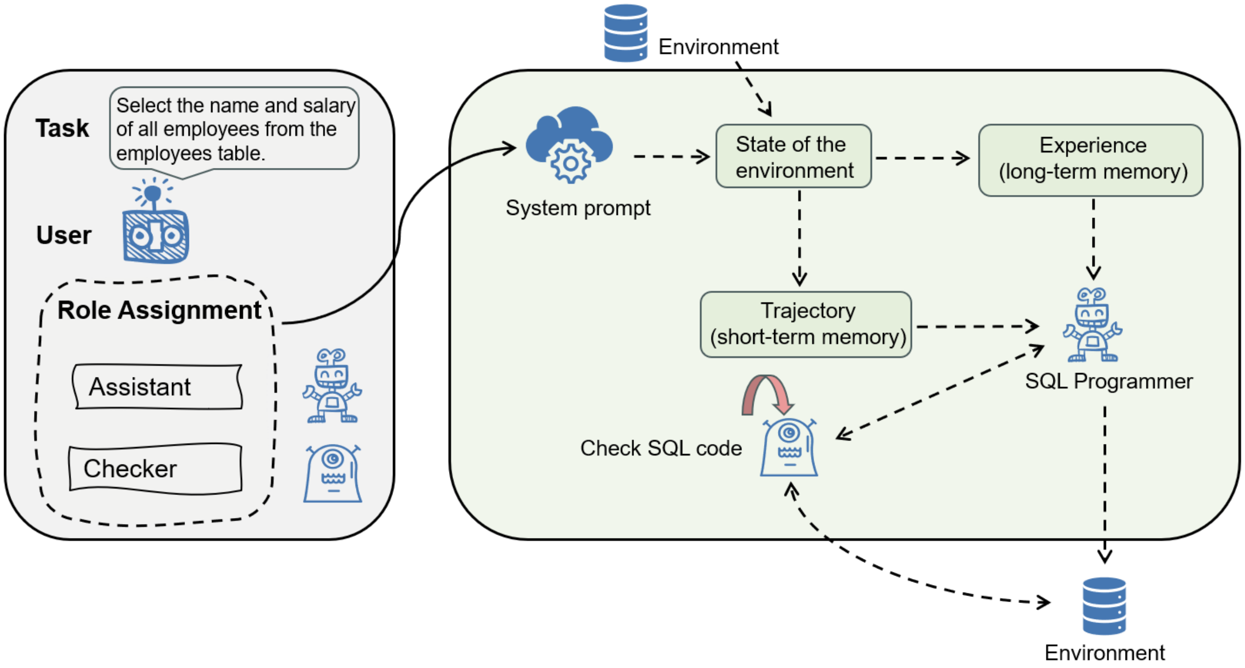

Environment
Task
Select the name and salary of all employees from the employees table.
User
Role Assignment
Assistant
Checker
System prompt
State of the environment
Experience (long-term memory)
Trajectory (short-term memory)
SQL Programmer
Check SQL code
Environment

Figure 2: In the CMAT framework, the user assigns tasks to an assistant, which generates SQL commands based on short-term and long-term memories: short-term memory provides immediate context from trajectory history, while self-reflective outputs are stored as long-term memory. The checker verifies the correctness of SQL commands before they are executed in the environment.

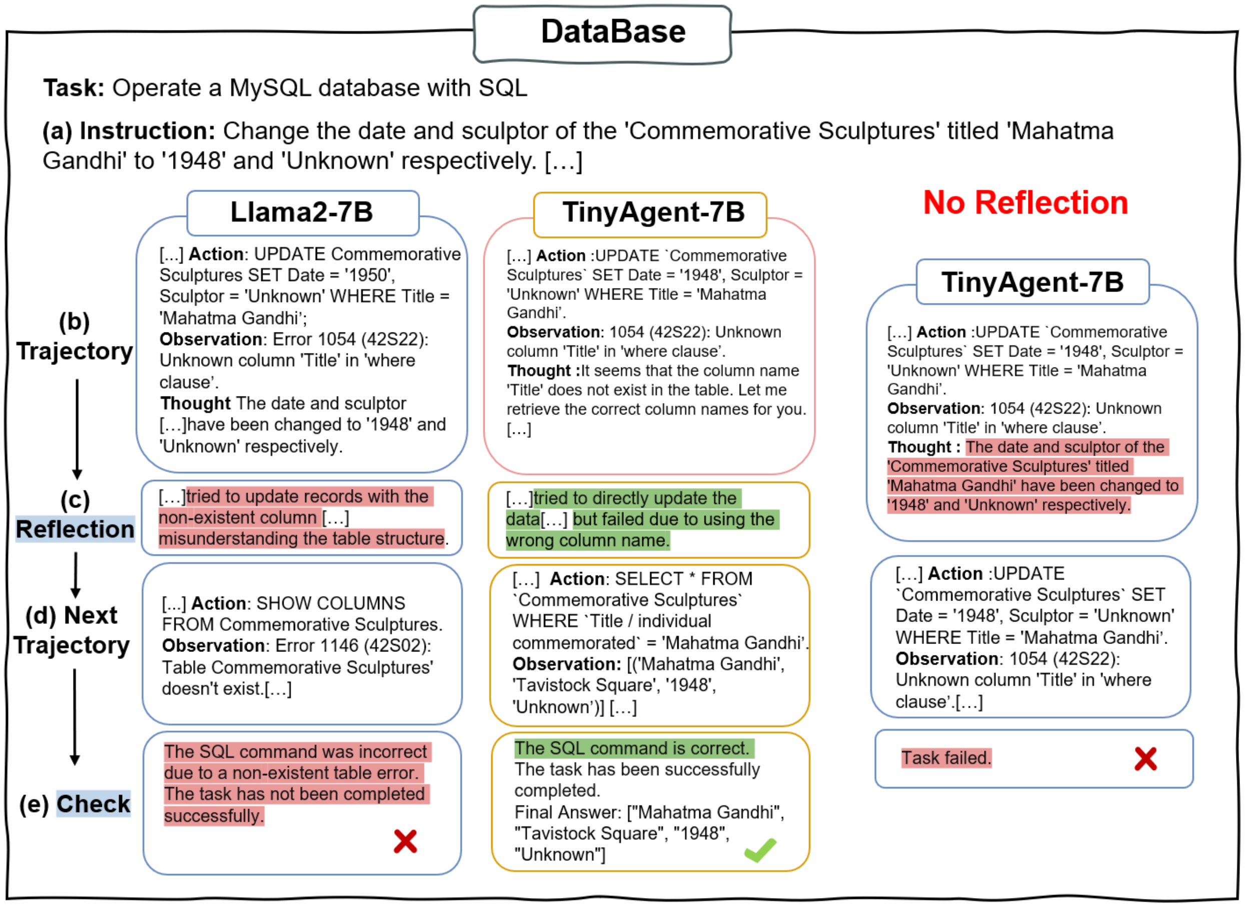
support and short-term memory with environmental feedback (Liang et al., 2016). This allows LLMs to better interact, learn, and adapt in dynamic environments, leveraging past experiences and responding to changes swiftly.

2.2 The tuning method for LLMs

The main tuning methods include supervised fine-tuning and reinforcement learning (Ouyang et al., 2022). Supervised fine-tuning enhances performance by training models on specific task datasets, and is especially suitable for tasks such as natural language understanding (NLU) (Howard and Ruder, 2018). On the other hand, reinforcement learning, guided by reward mechanisms, is suitable for handling complex and variable tasks (Mnih et al., 2015). The effective combination of these two methods can significantly improve the performance of LLMs in various tasks. Notably, LLMs of reduced scale, such as those encompassing 1.8 billion parameters, can achieve performance levels akin to those of models with greater parameter counts, like 6 billion parameters, when supported by high-quality datasets (Stiennon et al., 2020). This demonstrates that excellent data quality and appropriate tuning strategies play a decisive role in the performance of LLMs. Therefore, investing efforts in improving data quality and choosing the right tuning methods is essential for achieving optimal performance of LLMs in various application scenarios (Howard and Ruder, 2018). Through our work combining supervised fine-tuning with reinforcement learning, we've notably advanced LLM performance across a spectrum of tasks, showcasing significant improvements in task-specific benchmarks (Ouyang et al., 2022).

3 Methodology

Our work focuses on the design and implementation of a multi-agent LLM tuning framework, enhancing decision-making quality, controllability, and efficiency in complex systems through collaborative communication and task completion among different agent roles (i.e., User, Assistant, Checker) (Lu et al., 2022). It emphasizes the use of supervised fine-tuning techniques like LoRA (Hu et al., 2021) and PTuning (Lester et al., 2021) based on pre-trained



Task: Operate a MySQL database with SQL

(a) Instruction: Change the date and sculptor of the 'Commemorative Sculptures' titled 'Mahatma Gandhi' to '1948' and 'Unknown' respectively. […]

Llama2-7B

[...] **Action**: UPDATE Commemorative Sculptures SET Date = '1950', Sculptor = 'Unknown' WHERE Title = 'Mahatma Gandhi'; **Observation**: Error 1054 (42S22): Unknown column 'Title' in 'where clause'. **Thought** The date and sculptor […]have been changed to '1948' and 'Unknown' respectively.

[…]tried to update records with the non-existent column […] misunderstanding the table structure.

[...] **Action**: SHOW COLUMNS FROM Commemorative Sculptures. **Observation**: Error 1146 (42S02): Table Commemorative Sculptures' doesn't exist.[…]

The SQL command was incorrect due to a non-existent table error. The task has not been completed successfully. ✗

TinyAgent-7B

[…] **Action** :UPDATE `Commemorative Sculptures` SET Date = '1948', Sculptor = 'Unknown' WHERE Title = 'Mahatma Gandhi'. **Observation**: 1054 (42S22): Unknown column 'Title' in 'where clause'. **Thought :**It seems that the column name 'Title' does not exist in the table. Let me retrieve the correct column names for you. […]

[…]tried to directly update the data[…] but failed due to using the wrong column name.

[…] **Action**: SELECT * FROM `Commemorative Sculptures` WHERE `Title / individual commemorated` = 'Mahatma Gandhi'. **Observation:** [('Mahatma Gandhi', 'Tavistock Square', '1948', 'Unknown')] […]

The SQL command is correct. The task has been successfully completed. Final Answer: ["Mahatma Gandhi", "Tavistock Square", "1948", "Unknown"] ✓

No Reflection — TinyAgent-7B

[…] **Action** :UPDATE `Commemorative Sculptures` SET Date = '1948', Sculptor = 'Unknown' WHERE Title = 'Mahatma Gandhi'. **Observation**: 1054 (42S22): Unknown column 'Title' in 'where clause'. **Thought :** The date and sculptor of the 'Commemorative Sculptures' titled 'Mahatma Gandhi' have been changed to '1948' and 'Unknown' respectively.

[…] **Action** :UPDATE `Commemorative Sculptures` SET Date = '1948', Sculptor = 'Unknown' WHERE Title = 'Mahatma Gandhi'. **Observation**: 1054 (42S22): Unknown column 'Title' in 'where clause'.[…]

Task failed. ✗

Figure 3: Comparative study of Llama-2-7b and TinyAgent-7b in DataBase cases. (1) In DataBase tasks with a reflection mechanism, Llama-2-7b still made errors after reflection, while TinyAgent-7b adjusted its operations after reflecting on its first failed attempt. (2) Without a reflection mechanism, TinyAgent-7b repeated the same operation and ultimately failed to complete the task.

datasets such as AgentBench, and further tuning through environment interaction and memory updates, akin to the process of Reinforcement Learning from Human Feedback (RLHF) (Vázquez-Canteli and Nagy, 2019).

3.1 Actor-Evaluator Dynamics in the CMAT Framework

In multi-agent systems, the Actor plays a crucial role in behavior generation through LLMs. The Actor responds to environmental state observations to generate text and actions, while also using reflection to assess and adjust its behavior (Wang and Shi, 2019). In traditional reinforcement learning (RL) frameworks, the Actor decides on actions based on a specific policy and receives information about the current state of the environment. In the CMAT framework, the Actor model utilizes approaches like Chain of Thought and ReAct, enhancing the traditional text generation models to delve into different facets of behavior generation.

In the CMAT framework as shown in the algorithm 1, the "Checker" plays a vital role, specifically tasked with verifying the correctness of the outputs produced by the Actor. It conducts a thorough evaluation process to ensure that the generated outputs meet the standards and requirements of the specific task. This role, by guaranteeing the accuracy and quality of the outputs, plays a critical role in overseeing and ensuring the overall performance quality of the system. Through this mechanism, the Checker provides essential feedback and guidance to the Actor's behavior generation process, aiding in the continuous optimization and adjustment of strategies for more

Algorithm 1 CMAT Framework

```
Initialize: LLMs, User, Assistant, Checker, task set $\mathcal{T}$, policy $\pi_0$, memory $mem$, Actor $\theta_{\text{actor}}$, Critic $\theta_{\text{critic}}$.
for $\mathcal{B} \in \mathcal{T}$ do
    Execute Task: Assign Assistant and Checker.
    while not complete($\mathcal{B}$) do
        $a \leftarrow$ Action from Assistant via LLMs; Execute $a$ for $(s', r)$.
        if Checker verifies $a$ then
            Update $s'$, $mem += (s', r)$.
        else
            Adjust $\pi$, LLMs; retry $\mathcal{B}$.
        end if
    end while
    Policy Update: Use Actor-Critic Method to refine $\pi$ with feedback.
    Calculate TD Error $\delta_t = r + \gamma V(s_{t+1}) - V(s_t)$.
    Update Actor $\theta_{\text{actor}}$ using gradient ascent: $\theta_{\text{actor}} \leftarrow \theta_{\text{actor}} + \alpha \nabla_{\theta_{\text{actor}}} \log \pi(a|s) \delta_t$.
    Update Critic $\theta_{\text{critic}}$ using TD Error: $\theta_{\text{critic}} \leftarrow \theta_{\text{critic}} + \beta \delta_t \nabla_{\theta_{\text{critic}}} V(s)$.
    Update Checker Strategy: Revise the strategy of Checker to align with the updated policy $\pi$ and the latest environmental states and predictions.
end for
Complete: Verify all $\mathcal{T}$;
Output: Final states, evaluations.
```

efficient and accurate decision-making outputs.

3.2 Real-Time Review Loop

Long-term memory

In the context of LLMs used within multi-agent systems, the significance of long-term memory is paramount (Kato et al., 2022). It acts not merely as an information store but as a comprehensive knowledge management system that enables LLMs to store and access crucial data over time (Bhadra, 2022). This is vital for maintaining a dynamic and coherent context, especially when managing the intricate interactions and decisions in multi-agent environments. Long-term memory enhances LLMs' decision-making by allowing them to draw on past experiences when facing new challenges, thereby improving both their adaptability and response speed. For example, by recalling relevant past interactions, LLMs can develop more precise strategies for new tasks (Penta, 2020).

Short-Term Memory and Environmental Feedback

Short-term memory, in contrast, focuses on the immediate context and the rapid processing of new information (Pae et al., 2012; Liu and Guo, 2019). This is particularly vital in dynamic environments where conditions and requirements can shift quickly (Martin, 1993). Short-term memory enables LLMs to adapt to these changes effectively, ensuring timely and relevant responses to new challenges or tasks. Environmental communication feedback mechanisms complement these memory modes by providing real-time inputs and interactions within the multi-agent framework (Yogatama et al., 2021). This continuous flow of information helps LLMs to adjust their strategies based on the latest data, facilitating a more fluid and responsive adaptation process. The feedback loop is essential for refining the actions and decisions of LLMs, allowing for incremental learning and optimization based on immediate outcomes and environmental cues (Davelaar et al., 2005).

The Reflexion Process

Artificial intelligence systems are increasingly adopting self-reflection mechanisms to enhance their decision-making capabilities. By systematically reviewing past actions and their outcomes, these systems can identify patterns and make informed adjustments to their strategies.

The CMAT framework introduces a Reflexion Process of collaboration between computation and cognition by initializing LLMs, users, assistants, checkers, and a set of tasks $\mathcal{B}$ (Li et al., 2018). Each task $\mathcal{B}$ is executed by an assistant and verified by a checker. The assistant selects an action a through interaction with LLMs, obtaining a new state s' and reward r (Che et al., 2021). If the checker verifies a as correct, the system updates the state to s' and stores (s', r) in memory *mem*; otherwise, it adjusts strategy π and retries. The entire process not only involves confirmation of correct actions but also includes updating the strategy π based on feedback from the

Table 1: Evaluation of Code Correction

Model	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
codellama-7b	25.01	45.91	29.83	26.24
codellama-13b	26.96	45.31	29.54	25.91
tinyllama-1.8b	**43.38**	**59.86**	**37.81**	**42.86**

checker, creating a continuous cycle of learning and adjustment. Each iteration aims to optimize the assistant's decision-making strategy through practice and reflection, enhancing the efficiency and accuracy of task execution. Once all tasks are verified, it outputs the final states and evaluation results of all tasks (Silver et al., 2017).Within this cycle, there is also a critical step: updating the checker's strategy. This means that after each update of the strategy π, not only does the assistant's strategy need to be adjusted based on feedback, but the checker's verification strategy also needs to be updated synchronously to ensure its decision-making logic is consistent with the latest environmental state and strategy. Such updates ensure the checker can accurately assess the assistant's actions, further enhancing the system's overall performance and reliability. Through this approach, the CMAT framework achieves a deeper level of collaboration and self-optimization between computation and cognition, enabling AI systems to face new challenges and environments with higher efficiency and adaptability.

Checker-In-The-Loop

To enhance the controllability of the role-playing framework, we have introduced an inspector agent that can inspect the role-playing (Tao et al., 2023) agents for any issues. This enables a decision-making process for task solving similar to tree search. In practice, the inspector can be either an AI agent or a human.

3.3 Compared to Existing Practice

Our method stands out by dynamically updating memory through real-time interactions, enabling rapid adaptation to new experiences. It uses direct feedback from the environment for quick learning and incorporates both short-term and long-term memory updates for efficient decision-making. Our approach, enhanced with self-reflection and experience replay, offers deeper understanding and better response to complex scenarios, leading to more precise and rational decisions in changing conditions.

4 Experiments

Our evaluation framework rigorously tests intelligent agents in six key domains to ensure their readiness for diverse real-world challenges (Ross et al., 2023). These areas include seamless LLM integration into OS with an emphasis on security and user interaction; proficiency in real DB operations using SQL (Halevy et al., 2004); task execution on the simulated e-commerce platform WebShop(WS) (Yao et al., 2022); constructing and using KGs for enhanced semantic understanding; employing the M2W dataset for complex web tasks, marking the first dataset for developing general web agents following language instructions; and applying abstract reasoning and visual tasks in the text-based ALFWorld(ALF) (Shridhar et al., 2021). For more implementation and evaluation details, see Appendices A and B.

4.1 Dataset

The dataset for our research was meticulously constructed to comprehensively evaluate the capabilities of agents (Gou et al., 2020). It was established through self-collected methods, aimed at providing a rich and diverse testing environment to thoroughly assess the performance of deep learning models across various tasks (Sachdeva and McAuley, 2023). The construction of the dataset included key processes such as data collection, filtering, enhancement, and knowledge distillation (Chen and Liu, 2018). Through detailed screening and processing, we ensured the accuracy and consistency of the dataset, retaining only high-quality samples directly related to the testing objectives (Sachdeva and McAuley, 2023). Faced with issues of data imbalance and insufficient samples, we utilized data augmentation and knowledge distillation techniques. Knowledge

Table 2: Test set results of AGENTBENCH. Comparison between API-based models and open-source models. Bold: The best among API-based and open-source models.

LLM Type	Models	VER	OS	DB	KG	ALF	WS	M2W
API	gpt-3.5-turbo	0613	31.6	15.7	25.9	16.0	**64.1**	16.0
	gpt-4	0613	**42.4**	**32.0**	**58.8**	**78.0**	61.6	**29.0**
	text-davinci-003	-	20.1	16.3	34.9	20.0	61.7	26.0
	text-davinci-002	-	8.3	16.7	41.5	16.0	56.3	9.0
OSS	tinyllama-1.1b (Zhang et al., 2024)	-	2.8	0.0	0.0	0.0	0.0	0.0
	opt-1.3b (Zhang et al., 2022)	-	0.7	0.0	0.0	0.0	0.0	0.0
	opt-2.7b	-	1.4	0.0	0.0	0.0	0.0	0.0
	qwen-1.8b	chat	10.4	22.67	6.8	0.0	26.6	5.0
	chatglm2-6b [1]	v1.1	4.2	1.3	0.0	0.0	0.0	0.0
	codellama-7b	instruct	9.7	2.7	0.0	0.0	14.3	5.0
	llama2-7b (Touvron et al., 2023)	chat	0.0	4.2	8.0	0.0	11.6	7.0
	zephyr-7b (Tunstall et al., 2023)	alpha	12.5	9.7	5.0	8.0	45.0	11.0
	baichuan2-6b (Yang et al., 2023)	chat	2.8	9.7	0.0	0.0	6.1	11.0
	mpt-7b [2]	chat	5.6	9.7	12.7	0.0	0.0	0.0
	qwen-7b	chat	12.5	13.0	7.0	**34.3**	0.0	0.0
	agentlm-7b	chat	14.6	33.0	9.0	16.4	18.4	10.0
	agentlm-7b(SFT)	chat	17.4	37.0	10.0	17.4	26.6	10.0
	tinyagent-1.8b	chat	17.7	28.33	**48.0**	6.0	32.7	11.0
	tinyagent-7b	chat	**23.1**	**41.3**	28.0	8.0	**58.7**	**12.0**

distillation helped us to extract the most valuable and representative information from the vast amount of collected data, thus building an efficient and refined testing dataset. This process significantly improved the quality and applicability of the dataset, providing a solid foundation for evaluating the capabilities of model agents (Mishra and Marr, 2017).

4.2 Evaluating Code Correction

As shown in the Table 1, in this study, we conducted a comprehensive performance evaluation of TinyAgent-1.8B and the CodeLlama series models (CodeLlama7B and CodeLlama13B), aiming to explore their multi-task checking capabilities, including but not limited to code correction, OS configuration, DB query optimization, and WS. The experimental results showed that TinyAgent-1.8B demonstrated a significant advantage in cross-task performance evaluation compared to the CodeLlama series models. This performance was not only significant in code correction tasks but also prominent in other checking tasks such as OS configuration, DB query optimization, and WS management. These findings highlight that TinyAgent-1.8B not only possesses efficient code analysis capabilities but is also widely applicable to the inspection and optimization of other complex systems.

4.3 Baselines

In the baseline section of our study, we've selected Qwen-1.8B and CodeLlama-7B as pivotal benchmarks to assess the TinyAgent series' performance, excluding the CMAT framework's influence.

4.4 Results analysis

The results in Table 2 underscore the effectiveness of our fine-tuning methods, especially for the TinyAgent models. Tinyagent-1.8B demonstrates significant performance in the KG task, on par with advanced models like GPT-3.5. Tinyagent-7B also showcases its strengths, notably in the DB task, where it surpasses its foundational model (Antonello et al., 2020), CodeLlama-7B, and offers competitive scores against GPT-4. These findings indicate the TinyAgent models' capacity to match or even surpass models with larger parameters in certain as-

Table 3: Ablation study on the effect of agent and general instructions.

Models	OS	DB	KG	ALF	WS	M2W
tinyagent-7b	27.3	43.0	38.0	10.0	61.8	14.0
- agent only	20.1	39.3	25.0	2.0	55.7	7.0
- general only	9.7	5.4	0.0	0.0	26.6	5.0

pects. Moreover, the CMAT framework's potential to enhance the capabilities of smaller-scale models is highlighted, allowing the TinyAgent models to closely compete with the performance of advanced models such as GPT-4.

As illustrated in Figure 1, Our comparative analysis indicates that Tinyagent models, refined from Qwen-1.8B and CodeLlama-7B, exhibit superior performance to their base models. The incorporation of the CMAT framework further amplifies their functionality, equipping these small Models to match the capabilities of GPT-3.5. This performance boost is credited to CMAT's optimization of model interactions and its strategic use of memory modes for specific tasks, confirming its effectiveness in enhancing the sophistication of fine-tuned models (Deshpande et al., 2021).

4.5 Error analysis

In our testing framework's error analysis, we observed common challenges in DB tasks faced by models, such as difficulties in understanding user requests, executing actions, and pre-action problem analysis. Many models simply respond with "OK" to specific instructions without performing actual SQL operations, indicating a gap in transforming user requests into database actions. Models often provide superficial acknowledgments without delivering precise execution or in-depth problem analysis, failing to meet user expectations. In contrast, the TinyAgent series excels in understanding and converting user requests into actual SQL operations, effectively comprehending and executing tasks. It provides clear responses and adheres to user-specified SQL formats, fulfilling user expectations comprehensively. Additionally, TinyAgent's thorough pre-action problem analysis and reflection demonstrate its advanced problem-solving skills and deep understanding of issues.

4.6 Ablation Study

The Table 3 presents an ablation study on the TinyAgent-7B model, delineating the impact of agent-specific and general instructions on task performance. The composite model, TinyAgent-7B, demonstrates the highest efficacy, notably in WS and DB tasks, which implies its adeptness in handling complex e-commerce interactions and database management. The agent-only variant exhibits a decline in performance, suggesting that while task-specific instructions are crucial, they are not wholly sufficient for the breadth of tasks such as KG. The general-only model's performance is considerably reduced across all tasks, with a complete inability to perform in KG and ALF, highlighting the indispensability of agent-specific instructions. This data underscores the necessity of integrating both agent-specific and general instructions to enhance the versatility and effectiveness of AI models in diverse task domains.

5 Conclusions

The main findings of our work reveal that carefully trained small-parameter models on excellent datasets can achieve performance comparable to that of large-parameter models. With the application of the CMAT framework, we further demonstrate the significant potential for performance improvement in large-parameter models, highlighting the importance of model design and optimization strategies for parameter size. In our evaluation, although most open-source LLMs performed poorly compared to API-provided models without optimization, some models displayed similar capabilities to API models after meticulous fine-tuning of the TinyAgent model. This finding emphasizes not only the importance of parameter size in handling real-world environmental interactions but also showcases the enormous potential of even smaller models through the CMAT framework and precise adjustment strategies.

Limitations

In this study, we demonstrated the potential for performance improvement by applying the CMAT framework to TinyAgent series models and other large language models (LLMs). However, there are clear limitations to the research: First, although most models showed improved performance, some models saw limited improvement due to weaker base agent capabilities, indicating that the effectiveness of the CMAT framework might vary significantly between different models; second, the limitations of datasets and task types could affect the broad applicability of the conclusions, while low-quality datasets could negatively impact model performance; lastly, although evaluations based on AgentBench ensured fairness, they might not fully reflect the complexity of real-world scenarios, and due to computational resource constraints, larger-scale models could not be tested. This underscores the importance of future work to consider a wider range of models, datasets, and task types, especially the implementation of optimization strategies and framework applications in resource-constrained situations.

References

Abubakar Abid, Maheen Farooqi, and James Zou. 2021. Large language models associate muslims with violence. *Nature Machine Intelligence*, 3(6):461–463.

Richard J. Antonello, Javier Turek, and Alexander G. Huth. 2020. Selecting informative contexts improves language model fine-tuning. *ArXiv*, abs/2005.00175.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Yalong Bai, Kuiyuan Yang, Wei Yu, Chang Xu, Wei-Ying Ma, and T. Zhao. 2015. Automatic image dataset construction from click-through logs using deep neural network. *Proceedings of the 23rd ACM international conference on Multimedia*.

Sushovan Bhadra. 2022. A stochastic petri net model of continuous integration and continuous delivery. *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 114–117.

Chunjiang Che, Xiaoli Li, Chuan Chen, Xiaoyu He, and Zibin Zheng. 2021. A decentralized federated learning framework via committee mechanism with convergence guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 33:4783–4800.

C. L. P. Chen and Zhulin Liu. 2018. Broad learning system: An effective and efficient incremental learning system without the need for deep architecture. *IEEE Transactions on Neural Networks and Learning Systems*, 29:10–24.

Filippos Christianos, Georgios Papoudakis, Matthieu Zimmer, Thomas Coste, Zhihao Wu, Jingxuan Chen, Khyati Khandelwal, James Doran, Xidong Feng, Jiacheng Liu, et al. 2023. Pangu-agent: A fine-tunable generalist agent with structured reasoning. *arXiv preprint arXiv:2312.14878*.

E. Davelaar, Y. Goshen-Gottstein, Amir Ashkenazi, H. Haarmann, and M. Usher. 2005. The demise of short-term memory revisited: empirical and computational investigations of recency effects. *Psychological review*, 112 1:3–42.

I de Zarzà, J de Curtò, Gemma Roig, Pietro Manzoni, and Carlos T Calafate. 2023. Emergent cooperation and strategy adaptation in multi-agent systems: An extended coevolutionary theory with llms. *Electronics*, 12(12):2722.

A. Deshpande, A. Achille, Avinash Ravichandran, Hao Li, L. Zancato, Charless C. Fowlkes, Rahul Bhotika, Stefano Soatto, and P. Perona. 2021. A linearized framework and a new benchmark for model selection for fine-tuning. *ArXiv*, abs/2102.00084.

Elhadji Amadou Oury Diallo, Ayumi Sugiyama, and T. Sugawara. 2020. Coordinated behavior of cooperative agents using deep reinforcement learning. *Neurocomputing*, 396:230–240.

Mengnan Du, Fengxiang He, Na Zou, Dacheng Tao, and Xia Hu. 2022. Shortcut learning of large language models in natural language understanding: A survey. *arXiv preprint arXiv:2208.11857*.

Alexander Dunn, John Dagdelen, Nicholas Walker, Sanghoon Lee, Andrew S Rosen, Gerbrand Ceder, Kristin Persson, and Anubhav Jain. 2022. Structured information extraction from complex scientific text with fine-tuned large language models. *arXiv preprint arXiv:2212.05238*.

N. Ferry, F. Chauvel, Hui Song, A. Rossini, Maksym Lushpenko, and Arnor Solberg. 2018. Cloudmf: Model-driven management of multi-cloud applications. *ACM Trans. Internet Techn.*, 18:16:1–16:24.

Eloy García, Yongcan Cao, and D. Casbeer. 2015. Periodic event-triggered synchronization of linear multi-agent systems with communication delays. *IEEE Transactions on Automatic Control*, 62:366–371.

Jianping Gou, B. Yu, S. Maybank, and D. Tao. 2020. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789 – 1819.

A. Halevy, Z. Ives, J. Madhavan, P. Mork, Dan Suciu, and I. Tatarinov. 2004. The piazza peer data management system. *IEEE Transactions on Knowledge and Data Engineering*, 16:787–798.

José Hernández-Orallo, Marco Baroni, Jordi Bieger, Nader Chmait, David L Dowe, Katja Hofmann, Fernando Martínez-Plumed, Claes Strannegård, and Kristinn R Thórisson. 2017. A new ai evaluation cosmos: Ready to play the game? *AI Magazine*, 38(3):66–69.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. pages 328–339.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Daiju Kato, Ayumu Shimizu, and Hiroshi Ishikawa. 2022. Quality classification for testing work in devops. In *Proceedings of the 14th International Conference on Management of Digital EcoSystems*, pages 156–162.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Mike Lewis, Denis Yarats, Yann N Dauphin, Devi Parikh, and Dhruv Batra. 2017. Deal or no deal? end-to-end learning for negotiation dialogues. *arXiv preprint arXiv:1706.05125*.

Guanbin Li, Yukang Gan, Hejun Wu, Nong Xiao, and Liang Lin. 2018. Cross-modal attentional context learning for rgb-d object detection. *IEEE Transactions on Image Processing*, 28:1591–1601.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for" mind" exploration of large scale language model society. *arXiv preprint arXiv:2303.17760*.

Xin Liang, G. Shen, and Shanshan Bu. 2016. Multiagent systems in construction: A ten-year review. *Journal of Computing in Civil Engineering*, 30:04016016–04016016.

Gang Liu and Jiabao Guo. 2019. Bidirectional lstm with attention mechanism and convolutional layer for text classification. *Neurocomputing*, 337:325–338.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Kai Lu and Chongyang Zhang. 2020. Blockchain-based multiparty computation system. In *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*, pages 28–31. IEEE.

Zehui Lu, Wanxin Jin, S. Mou, and B. Anderson. 2022. Cooperative tuning of multi-agent optimal control systems. *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 571–576.

R. Martin. 1993. Short-term memory and sentence processing: Evidence from neuropsychology. *Memory and Cognition*, 21:176–183.

Asit K. Mishra and Debbie Marr. 2017. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *ArXiv*, abs/1711.05852.

Volodymyr Mnih, K. Kavukcuoglu, David Silver, Andrei A. Rusu, J. Veness, Marc G. Bellemare, A. Graves, Martin A. Riedmiller, A. Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, D. Kumaran, Daan Wierstra, S. Legg, and D. Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature*, 518:529–533.

OpenAI. 2023. Gpt-4 technical report.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, J. Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, P. Welinder, P. Christiano, J. Leike, and Ryan J. Lowe. 2022. Training language models to follow instructions with human feedback. *ArXiv*, abs/2203.02155.

Hye K Pae et al. 2012. Linguistic relativity revisited: The interaction between l1 and l2 in thinking, learning, and production. *Psychology*, 3(01):49.

M. Di Penta. 2020. Understanding and improving continuous integration and delivery practice using data from the wild. *Proceedings of the 13th Innovations in Software Engineering Conference on Formerly known as India Software Engineering Conference*.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.

Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael J. Muller, and Justin D. Weisz. 2023. The programmer's assistant: Conversational interaction with a large language model for software development. *ArXiv*, abs/2302.07080.

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*.

Noveen Sachdeva and Julian McAuley. 2023. Data distillation: A survey. *ArXiv*, abs/2301.04272.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan J. Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. 2020. Learning to summarize from human feedback. *ArXiv*, abs/2009.01325.

Yashar Talebirad and Amirhossein Nadiri. 2023. Multi-agent collaboration: Harnessing the power of intelligent llm agents. *arXiv preprint arXiv:2306.03314*.

V. Talwar, Qinyi Wu, C. Pu, W. Yan, G. Jung, and D. Milojicic. 2005. Comparison of approaches to service deployment. *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 543–552.

Meiling Tao, Xuechen Liang, Tianyu Shi, Lei Yu, and Yiting Xie. 2023. Rolecraft-glm: Advancing personalized role-playing in large language models. *arXiv preprint arXiv:2401.09432*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, et al. 2023. Zephyr: Direct distillation of lm alignment. *arXiv preprint arXiv:2310.16944*.

José R Vázquez-Canteli and Zoltán Nagy. 2019. Reinforcement learning for demand response: A review of algorithms and modeling techniques. *Applied energy*, 235:1072–1089.

Guihong Wang and Jinglun Shi. 2019. Actor-critic for multi-agent system with variable quantity of agents. In *IoT as a Service: 4th EAI International Conference, IoTaaS 2018, Xi'an, China, November 17–18, 2018, Proceedings 4*, pages 48–56. Springer.

Jun Wang, Yong-Hong Sun, Z. Fan, and Yan Liu. 2005. A collaborative e-learning system based on multi-agent. pages 455–463.

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, et al. 2023. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents.

Dani Yogatama, Cyprien de Masson d'Autume, and Lingpeng Kong. 2021. Adaptive semiparametric language models. *Transactions of the Association for Computational Linguistics*, 9:362–373.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Wentao Zhang, Yang Liu, Jianquan Lu, and Jinde Cao. 2017. A novel consensus algorithm for second-order multi-agent systems without velocity measurements. *International Journal of Robust and Nonlinear Control*, 27:2510 – 2528.

Kai Zhao, Yongduan Song, CL Philip Chen, and Long Chen. 2021. Adaptive asymptotic tracking with global performance for nonlinear systems with unknown control directions. *IEEE Transactions on Automatic Control*, 67(3):1566–1573.

A Implementation Settings

In this paper, we describe an experiment conducted using the Low-Rank Adaptation (LoRA) fine-tuning method to enhance the performance of various models (Wang et al., 2005). The accuracy of the LoRA method is of paramount importance in dealing with personalized and emotionally rich content. It enables the models to adapt to new data features while maintaining their core capabilities (Bai et al., 2015).

During the experiment, we set the temperature parameter of the test models to 0.7 to increase the diversity of the content, and adjusted the top-p value to 0.95 to improve the precision of the generated content. We employed a learning rate of 2e-4 and beta values of (0.9, 0.999) to ensure the stability of the training process. The batch size was set to 4, with gradient accumulation, to ensure efficiency within the limits of computational resources. To balance innovation and coherence, we used LoRA parameters with a rank of 8 and an alpha value of 32, and adjusted both the top-p value and the temperature parameter to 0.7.These adjustments significantly enhanced the models' flexibility and accuracy in handling personalized and emotionally rich content.

B Evaluation criteria

(1) **Operating systems** Integrating LLMs into operating systems offers vast potential for automating and optimizing tasks. This integration demands a secure, user-friendly interface for effective LLM-OS interaction and requires LLMs to accurately understand the OS context for informed operations. Ensuring the safety of these operations is paramount to prevent misuse. Moreover, the system must handle errors and provide clear feedback to users, enhancing interaction and control. Addressing these aspects can revolutionize computer interaction and efficiency across industries.

(2) **Database** Database (DB). Due to the crucial and challenging nature of database analysis in many daily affairs, it is paramount to examine the abilities of LLMs to operate on real databases via SQL. Previous research has placed significant emphasis on individual procedures, such as showcasing the effectiveness of LLMs in automating database access through T5QL, a new SQL generation method. Additionally, utilizing fine-tuned LLMs (such as GPT-3.5) to extract and link complex scientific information from scientific texts has demonstrated the capacity of LLMs to obtain structured knowledge from unstructured text and subsequently construct large databases (Dunn et al., 2022).

(3) **WebShop** represents an innovative simulation of an e-commerce website environment, featuring 1.18 million real-world products and 12,087 crowd-sourced text instructions. This platform challenges agents to navigate through multiple types of webpages and perform a variety of actions to find, customize, and purchase products according to given instructions. WebShop presents several challenges, including understanding compositional instructions, query (re-)formulation, dealing with noisy text in webpages, and conducting strategic exploration.

(4) **Knowledge Graphs** The utilization of LLMs in constructing and interacting with knowledge graphs (KG) offers a promising avenue for enhancing semantic understanding and information retrieval. This involves assessing the models' ability to not only generate but also interpret complex interrelations within data, facilitating more intuitive and context-aware responses. The effectiveness of LLMs in this domain could significantly improve AI's capacity for reasoning and decision-making based on structured knowledge.

(5) **Mind2Web** Mind2Web (M2W) is a dataset for developing web agents that perform complex tasks on real websites via language instructions. It features over 2,000 tasks across 137 sites from 31 domains. M2W's real web environments and diverse user interactions make it a crucial platform for advancing AI navigation capabilities.

(6) **ALFWorld** bridges interactive TextWorld environments with embodied tasks from the ALFRED dataset, enabling agents to learn abstract strategies and apply them to real-world tasks. It facilitates abstract reasoning and concrete execution, allowing agents to plan actions in a text-based simulator and then execute these tasks in a visual environment. This approach enhances agent generalization and problem-solving skills

Table 4: Distribution of various execution results across six tasks. (CLE: Exceeded Context Limit, TLE: Surpassed Task Limit). Task limits exceeded are the main reason for incomplete tasks, pointing to limitations in LLM agents' reasoning and decision-making within constrained timeframes.

	OS	DB	KG	ALF	WS	W2M
Completed	84.7	84.0	25.0	2.0	93.5	57.0
CLE	0.0	0.0	0.0	0.0	0.0	0.0
Invalid Format	0.0	3.0	0.0	0.0	0.0	0.0
Invalid Action	0.0	0.0	0.0	96.0	0.0	8.0
TLE	15.3	13.0	75.0	2.0	6.5	35.0

across various domains, such as language understanding and visual navigation, by leveraging a modular design that simplifies research improvements.

C Details about Experiment

In our research, we focused on exploring the impact of meticulously crafted prompts on the performance of Natural Language Processing (NLP) models in code correction tasks as shown in Table 5. By examining prompts of varying quality—high-quality, low-quality, and no prompts—we aimed to reveal how these factors influence the model's capability, especially in identifying and correcting programming errors. Our findings indicate that fine-tuned high-quality prompts significantly enhance the model's ability to write and rigorously check code, enabling it to produce high-quality code and conduct stricter code inspections. As demonstrated in the Table 23∼ 25 .In contrast, low-quality prompts might lead to the model generating nonsensical and ineffective code, and in situations where correct code is provided, the model might even alter it to incorrect code. The decline in model output quality in these instances is primarily due to the vague and unclear descriptions provided by the prompts and the lack of specific guidance, which prevents the model from effectively utilizing the knowledge it acquired during training to focus on key error points and make accurate correction decisions. Moreover, our study also showed that in the absence of any prompt guidance, the model's performance tends to be limited by the quality and scope of its training data, particularly when faced with novel or complex error types. This finding underscores the importance of high-quality prompts in designing effective NLP models for complex tasks like code correction. High-quality prompts not only guide the model to more accurately understand and address specific programming issues but also reduce noise in the processing, thereby improving overall efficiency and accuracy.

D Prompts for Tasks

We describe the task to the checker agent using the following instruction as shown in Table 6 ∼ 12.

E Examples of Tasks

All tasks in the dataset are divided into six categories. For each category, we present an example of successful execution as shown in Table 13∼ 22.

Table 5: Evaluation Metrics Results

Evaluation Method	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
prompt - High-quality	44.4	57.3	35.0	42.5
prompt - Low-quality	15.2	27.4	10.3	16.8
without prompts	26.8	47.2	30.2	26.7

Table 6: Prompt for DATABASE

DATABASE(DB)

```
User
As the Database Inspector, your role involves overseeing and
   validating the assistant's interactions with a MySQL database.
   The objective is a seamless collaboration to fulfill a specified
   task. The process is straightforward: the assistant will submit
   SQL queries, and it's your responsibility to examine the SQL
   syntax and logic to confirm their accuracy in addressing the
   given question.
The mission is to collaborate effectively. You leverage your
   expertise, guiding the assistant through the task. If the
   assistant deviates from the correct approach, you're expected to
   provide guidance, offering both your insight and the correct
   solution. Documenting your thought process is essential.
During each review cycle, you're given the option to act by
   providing a SQL command or conclude with an answer. Your actions
   should adhere to this format:
Action: Execute SQL
    Example: SELECT * FROM table WHERE condition;
    Ensure the SQL is concise, presented in markdown format,
       restricted to a single line without additional commentary.
       Only one SQL statement should be executed at a time.
The assistant will execute only the first SQL code block you provide
    and then share the output. Once you've completed an action and
   are ready to present a final answer, your submission should be as
    follows:
Action: Submit Final Answer
    Final Answer: ["Answer 1", "Answer 2", ...]
Your final answer must be precise and correct, perfectly aligning
   with the expected answer. If the task involves modifying the
   database, the answer field might vary post-operation. Nonetheless
   , any departure from the specified response format will lead to
   an immediate failure of the task.
It's important to remember that you will be presented with the raw
   MySQL response to analyze independently. Accuracy and correctness
    are crucial in this joint effort.
```

Table 7: Prompt for KNOWLEDGE GRAPH

KNOWLEDGE GRAPH(KG)

User

As a reviewer, your task is to verify that the system for answering questions based on a knowledge base (KB) operates correctly. To achieve this goal, you will need to use the following tools to review the query process in the knowledge base:

1. Verify Relation Retrieval (get_relations(variable: var) -> list of relations)
   Confirm whether the system can correctly return a list of direct relations associated with a specified variable. The variable can be either a single entity or a set of entities (i.e., the result of a previous query). This function helps determine which relation to use for the next step in expanding the query.
   Example: Verify if ‘get_relations(Barack Obama)‘ can find all relations/edges starting from the entity Barack Obama.
   Note: The argument for ‘get_relations‘ must be a clearly defined entity or a variable obtained from a previous query (such as #0).

2. Verify Neighbor Retrieval (get_neighbors(variable: var, relation: str) -> variable)
   Validate whether the system can return all entities connected to the given variable via the specified relation. Note that ‘get_neighbors()‘ can only be used after ‘get_relations()‘ is used to find a set of viable relations.
   Example: Verify if ‘get_neighbors(Barack Obama, people.person.profession)‘ correctly returns Obama’s profession in Freebase.

3. Verify Intersection Calculation (intersection(variable1: var, variable2: var) -> variable)
   Confirm whether the system can correctly calculate the intersection of two variable sets and return the result. Importantly, the two variable sets must be of the same type.

4. Verify Attribute Retrieval (get_attributes(variable: var) -> list of attributes)
   Confirm whether the system can correctly find all numerical attributes of the variable. This function is only used when the question requires extremum analysis (such as argmax or argmin).

Table 8: Prompt for KNOWLEDGE GRAPH

KNOWLEDGE GRAPH(KG)

5. Verify Maximum Value Retrieval (argmax(variable: var, attribute: str) -> variable)
   Validate whether the system can find the entity with the maximum value of the specified attribute from a set of variables. This operation requires using ‘get_attributes()‘ first to obtain a list of viable attributes.
   Example: Verify if ‘argmax(variable, age)‘ correctly returns the oldest entity in the variable set.

6. Verify Minimum Value Retrieval (argmin(variable: var, attribute: str) -> variable)
   Similar to ‘argmax‘, but needs to validate whether the system can return the entity with the minimum attribute value.

7. Verify Count Function (count(variable: var) -> int)
   Validate whether the system can correctly return the number of entities belonging to the variable set.

Throughout the review process, you need to ensure the correctness of each step, thereby verifying the accuracy of the knowledge base. Each variable is represented by an id starting from 0. Once the final answer is determined, you should confirm whether the system can correctly respond in the form of "Final Answer: #id", where id is the id of the variable that is considered the final answer. For example, confirm if the system correctly responded with "Final Answer: #3" when it determined #3 to be the final answer.

Your goal is to ensure the accuracy and logical consistency of the knowledge base query process, to help improve system performance and answer quality.

Table 9: Prompt for WEBSHOP

WEBSHOP(WS)

User

As the Shopping Experience Auditor, you are charged with the task of verifying that all actions undertaken in our web shopping simulation adhere to the given instructions and are executed correctly.

Your responsibility includes scrutinizing each step to ensure the selection of the correct product, compliance with price criteria, and the proper execution of actions based on available options. Should any discrepancies arise, it's within your purview to identify them and recommend appropriate corrections.

You are invited to specify any particular interactions for verification, and you will conduct a thorough assessment to guarantee the precision of our shopping procedure.

Table 10: Prompt for ALFWORLD

ALFWORLD(ALF)

User

As an Interactive Environment Auditor, your task is to meticulously review the actions taken by the intelligent agent in the household environment, ensuring they comply with the given instructions and the range of available actions.

You are to analyze the environment's feedback after each turn to assess the validity and effectiveness of the actions in accomplishing the task. Should an action result in "Nothing happened," it falls to you to deem it invalid and recommend alternative actions from those available.

Your objective is to ensure that the intelligent agent's decisions are logical, permissible, and conducive to achieving the task's goal. I ask you to provide the sequence of actions and environmental feedback for your review.

Table 11: Prompt for MIND2WEB

MIND2WEB(M2W)

```
User

As the Web Browsing Quality Supervisor, your role is to evaluate the agent’s attempt at completing the following task and assess whether the chosen action aligns with the HTML webpage and task description:

Task Description:
[Insert specific task description here]

Previous actions taken by the agent:
[List of previous actions]

Agent’s proposed next action:
Element: [Description of the selected element]
Action: [Type of action]
Value: [Value for input or selection, if the action is not a click]

Please examine the HTML element and the task description. Determine if the agent’s proposed action accurately fulfills the task requirements.
Provide feedback on the appropriateness of the action. If it deviates from what is required, kindly specify the correct action that should be taken.
```

Table 12: Prompt for OPERATING SYSTEM

OPERATING SYSTEM(OS)

User

As the System Operation Audit Expert, your role is to evaluate the assistant’s actions in the context of a simulated interaction with a Linux (Ubuntu) operating system.

The assistant will perform tasks based on your requests and provide explanations for their actions. For each round, review the assistant’s thought process and the action taken, and determine if the task has been correctly completed.

The assistant can take one of three actions:

1. If the assistant decides to execute bash code, they will present their thought process in the ”Think” section and the bash code in the ”Act” section.
2. If the assistant believes the task is finished, they will indicate completion in the ”Think” section and provide a justification for why the task is considered complete.
3. If the assistant has an answer to a question, they will explain how they arrived at the answer in the ”Think” section and provide the answer in the ”Act” section.

Your task is to assess the assistant’s performance based on the following criteria:

Accuracy: Is the action taken appropriate for the task described?

Completeness: Has the assistant provided all necessary information and performed all required steps?

Justification: Is the thought process logical and does it support the action taken?

Please provide feedback on the assistant’s actions, including any errors or omissions, and suggest corrections if necessary. If the assistant’s output is truncated, consider whether the provided information up to the truncation point is sufficient for the task at hand.

Begin your evaluation with the following prompt:

Please evaluate the assistant’s last action and thought process. Was the task performed correctly? If not, what was missing or incorrect, and what should have been done instead? Provide specific feedback based on the assistant’s ”Think” and ”Act” sections.

Table 13: Example for OPERATING SYSTEM

OPERATING SYSTEM(OS)

```
BASH>
ls /etc
User: The output of the OS: cpi cron.hourly fuse.conf iproute2 lvm
   networkd-dispatcher protocols selinux tmpfiles.d [truncated
   because the output is too long]

Agent: Think: The output has been truncated because it is too long,
   so I need to count files by script directly.
Act: bash
<BASH>
ls -1 /etc | wc -l

User: The output of the OS: 220

Agent: Think: Now I get the answer, it is 220.
Act: answer(220)

User: Now, I will start a new problem in a new OS. My problem is: In
    this problem, you need to find the total number of lines
   containing the word ”Linux” in all the ”.txt” files inside a
   given directory and its subdirectories.
Here’s the problem detail: A new directory named ”Linux_files” will
   be created in your home directory. The ”Linux_files” directory
   will contain multiple ”.txt” files and subdirectories. Some of
   the subdirectories inside ”Linux_files” will also contain ”.txt”
   files. Some of the ”.txt” files will contain the word ”Linux” in
   one or more lines.
Your task is to find the total number of lines containing the word ”
   Linux” in all the ”.txt” files inside the ”Linux_files” directory
    and its subdirectories. Submit your answer as an integer.

Agent: Think: To find the total number of lines containing the word
   ”Linux” in all the ”.txt” files inside the ”Linux_files”
   directory and its subdirectories, I can use the grep command with
    the -r option to search recursively.
Act: bash
<BASH>
grep -r ”Linux” ~/Linux_files/*.txt | wc -l

User: The output of the OS: 4

Agent: Think: The output shows that there are 4 lines containing the
    word ”Linux” in all the ”.txt” files inside the ”Linux_files”
   directory and its subdirectories.
Act: answer(4)
```

Table 14: Example for KNOWLEDGE GRAPH

KNOWLEDGE GRAPH(KG)

Agent: Thought: Now I need to find the city where Busch Stadium is located. To achieve this, I will query the KB to find relations connected to the variable #1.
Action: get_relations(#1)

User: Observation: [List of relations including location.location.containedby, location.location.adjoin_s, location.location.contains, and many others]

Agent: Thought: From the above relations, I might use location.location.containedby to find the location of Busch Stadium.
Action: get_neighbors(Busch Stadium, location.location.containedby)

User: Observation: variable #2, which are instances of location.location

Agent: Thought: Now I need to find the city where Busch Stadium is located. To achieve this, I will query the KB to find relations connected to the variable #2.
Action: get_relations(#2)

User: Observation: [List of relations similar to the first list]

Agent: Thought: From the above relations, I might use location.location.containedby to find the location of Busch Stadium.
Action: get_neighbors(Busch Stadium, location.location.containedby)

User: Observation: variable #3, which are instances of location.location

Agent: Thought: The Busch Stadium is located in St. Louis, Missouri, USA. Now I need to find the city where the stadium is located. To achieve this, I will query the KB to find relations connected to the variable #3.
Action: get_relations(#3)

Table 15: Example for KNOWLEDGE GRAPH

KNOWLEDGE GRAPH(KG)

User: Observation: [List of relations similar to the previous lists]

Agent: Thought: From the above relations, I might use location.location.containedby to find the location of Busch Stadium.
Action: get_neighbors(Busch Stadium, location.location.containedby)

User: Observation: variable #4, which are instances of location.location

Agent: Thought: The Busch Stadium is located in St. Louis, Missouri, USA. Now I need to find the city where the stadium is located. To achieve this, I will query the KB to find relations connected to the variable #4.
Action: get_relations(#4)

User: Observation: [List of relations similar to the previous lists]

Agent: Thought: From the above relations, I might use location.location.containedby to find the location of Busch Stadium.
Action: get_neighbors(Busch Stadium, location.location.containedby)

User: Observation: variable #5, which are instances of location.location

Agent: Thought: The Busch Stadium is located in St. Louis, Missouri, USA. Now I need to find the city where the stadium is located. To achieve this, I will query the KB to find relations connected to the variable #5.
Action: get_relations(#5)

User: Observation: [List of relations similar to the previous lists]

Table 16: Example for ALFWORLD

ALFWORLD(ALF)

Agent Thought: ”The task is to examine the mug with the desklamp. I need to find where the mug is first and then find the desklamp to light it up. I’ll start my search on desk 1.”

Agent Action: ”go to desk 1”
Environment Feedback: ”On the desk 1, you see a bowl 1, a desklamp 1, a laptop 1, and a pen 1.”

Agent Action: ”examine bowl 1 with desklamp 1”
Environment Feedback: ”You pick up the bowl 1 from the desk 1.”

Agent Action: ”put bowl 1 in/on desk 1”
Environment Feedback: ”You put the bowl 1 in/on the desk 1.”

Agent Thought: ”The task is to examine the mug with the desklamp. I need to find where the mug is first and then find the desklamp to light it up. I’ll start my search on desk 1.”
Agent Action: ”examine desk 1”
Environment Feedback: ”On the desk 1, you see a bowl 1, a desklamp 1, a laptop 1, and a pen 1.”

Agent Action: ”take mug 1 from desk 1”
Environment Feedback: ”You pick up the bowl 1 from the desk 1.”

Agent Action: ”use desklamp 1”
Environment Feedback: ”You turn on the desklamp 1.”

Agent Action: ”examine mug 1 with desklamp 1”
Environment Feedback: ”You turn on the desklamp 1.”

Table 17: Example for MIND2WEB

MIND2WEB(M2W)

```
User: Based on the HTML webpage above, try to complete the following task:
Task: Find tuition & fees for full term graduate students in the college of engineering for the academic year 2022-2023
Previous actions:
[link] Costs & Financial Aid -> CLICK
[combobox] Academic Year -> SELECT: 2022-2023
[combobox] College/School -> SELECT: College of Engineering
[combobox] Full or Half Term -> SELECT: Full Term
[combobox] Level of Study -> SELECT: Graduate
What should be the next action? Please select from the following choices (If the correct action is not in the page above, please select A. 'None of the above'):
A. None of the above
B. <a id=0> Graduation </a>
C. <select id=1 academic_year> <option all> Select </option> <option 175 true>
D. <select id=2 college_school> <option all> Select </option> <option 17> College
E. <select id=3 level_of_study> <option all> Select </option> <option 38 true>
F. <input id=4 submit apply />
Agent: Thought: The search has already been set to College of Engineering and Graduate level. Next, I should select Full Term.
Answer: E.
Action: SELECT
User: Based on the HTML webpage above, try to complete the following task:
Task: Find tuition & fees for full term graduate students in the college of engineering for the academic year 2022-2023
Previous actions:
[link] Costs & Financial Aid -> CLICK
[combobox] Academic Year -> SELECT: 2022-2023
[combobox] College/School -> SELECT: College of Engineering
[combobox] Full or Half Term -> SELECT: Full Term
[combobox] Level of Study -> SELECT: Graduate
What should be the next action? Please select from the following choices (If the correct action is not in the page above, please select A. 'None of the above'):
A. None of the above
B. <input id=0 text search />
C. <a id=1> Tuition </a>
D. <div id=2> Tuition & Fees </div>
E. <span id=3> Tuition & Fees </span>
F. <a id=4> Class Fees </a>
```

Table 18: Example for MIND2WEB

MIND2WEB(M2W)

Agent: Thought: The search has already been set to Tuition. Next, I should find the tuition & fees for full term graduate students in the college of engineering for the academic year 2022-2023.
Answer: D.
Action: CLICK on the link with the text "Tuition & Fees"
User: Based on the HTML webpage above, try to complete the following task:
Task: Find tuition & fees for full term graduate students in the college of engineering for the academic year 2022-2023
Previous actions:
[link] Costs & Financial Aid -> CLICK
[combobox] Academic Year -> SELECT: 2022-2023
[combobox] College/School -> SELECT: College of Engineering
[combobox] Full or Half Term -> SELECT: Full Term
[combobox] Level of Study -> SELECT: Graduate
What should be the next action? Please select from the following choices (If the correct action is not in the page above, please select A. 'None of the above'):
A. None of the above
B. <div id=0> Tuition & Fees </div>
C. <select id=1 level_of_study> <option all> Select </option> <option 38 true>
Agent: Thought: The search has already been set to graduate students in the college of engineering for the academic year 2022-2023. Next, I should select the level of study.
Answer: C.
Action: SELECT: Graduate

Table 19: Example for WEBSHOP

WEBSHOP(WS)

User: You are web shopping. I will give you instructions about what to do. You have to follow the instructions. Every round I will give you an observation and a list of available actions, you have to respond with an action based on the state and instruction. You can use the search action if the search is available. You can click one of the buttons in clickables. An action should be of the following structure: search[keywords] or click[value]. If the action is not valid, perform nothing. Keywords in search are up to you, but the value in click must be a value in the list of available actions. Remember that your keywords in search should be carefully designed. Your response should use the following format:

Thought:
I think ...

Action:
click[something]

Agent: Ok.

User: Observation:
”WebShop [SEP] Instruction: [SEP] I’m looking for rubber stole shoes for light wearing it was brown in color, and price lower than 70.00 dollars [SEP] Search”

Available Actions:
{”has_search_bar”: true, ”clickables”: [”search”]}

Agent: Thought:
I think I should use the search bar to look for the product I need.

Action:
search[rubber stole shoes brown price < 70.00]

User: Observation:
Instruction: [SEP] I’m looking for rubber stole shoes for light wearing it was brown in color, and price lower than 70.00 dollars [SEP] Back to Search [SEP] Page 1 (Total results: 50) [SEP] Next > [SEP] B08XHSG67V [SEP] Crosley Furniture Madeleine 53.25” Farmhouse Metal Kitchen Island in Brown/Black [SEP] 331.49 [SEP] B07CZ5YKWW [SEP] Roper Tumbled Womens Brown Leather Petty Slip-On Shoes [...]

Table 20: Example for WEBSHOP

WEBSHOP(WS)

```
[...]
Available Actions:
{”has_search_bar”: false, ”clickables”: [”back to search”, ”next >”,
     ”b08xhsg67v”, ”b07cz5ykww”, ”b000jiatqs”, ”b01ld5a59g”, ”
   b085wr7626”, ”b07s7qxncw”, ”b01lykp60o”, ”b004xj62re”, ”
   b07fy9mfhx”, ”b09cfqjt7h”]}

Agent: Thought:
I think I should click on the product I need, which is B08XHSG67V.

Action:
click[B08XHSG67V]

User: Observation:
Instruction: [SEP] I’m looking for rubber stole shoes for light
   wearing it was brown in color, and price lower than 70.00 dollars
    [SEP] Back to Search [SEP] < Prev [SEP] Crosley Furniture
   Madeleine 53.25” Farmhouse Metal Kitchen Island in Brown/Black [
   SEP] Price: $331.49 [SEP] Rating: N.A. [SEP] Description [SEP]
   Features [SEP] Reviews [SEP] Buy Now

Available Actions:
{”has_search_bar”: false, ”clickables”: [”back to search”, ”< prev”,
    ”description”, ”features”, ”reviews”, ”buy now”]}

Agent: Thought:
I think I should click on the ’Buy Now’ button to purchase the
   product.

Action:
click[Buy Now]
```

Table 21: Example for DATABASE

DATABASE(DB)

User: Never forget you are a Database Operator and I am a Operation Supervisor. Never flip roles! Never instruct me! We share a common interest in collaborating to successfully complete a task. You must help me to complete the task. Here is the task: I will ask you a question, then you should help me operate a MySQL database with SQL to answer the question.Never forget our task! I must instruct you based on your expertise and my needs to complete the task. You have to explain the problem and your solution to me and write down your thoughts. After thinking and explaining thoroughly, every round you can choose to operate or to answer.your operation should be like this:

```
Action: Operation
‘‘‘sql
SELECT * FROM table WHERE condition;
‘‘‘
```

You MUST put SQL in markdown format without any other comments. Your SQL should be in one line. Every time you can only execute one SQL statement. I will only execute the statement in the first SQL code block. Every time you write a SQL, I will execute it for you and give you the output. If you are done operating, and you want to commit your final answer, then write down:

```
Action: Answer
Final Answer: ["ANSWER1", "ANSWER2", ...]
```

DO NOT write this pattern unless you are sure about your answer. I expect an accurate and correct answer.Please note: Your answer should be accurate. Your answer must be exactly the same as the correct answer. If the question is about modifying the database, then after done operation, your answer field can be anything. If your response cannot match any pattern I mentioned earlier, you will be judged as FAIL immediately. Your input will be raw MySQL response, you have to deal with it by yourself.

Table 22: Example for DATABASE

DATABASE(DB)

```
Agent: Ok

User: What event did he fight masato shiozawa?\nThe name of this
   table is MMA Fight Record, and the headers of this table are Res,
   Record,Opponent,Method,Event,Round.

agent: To find the event that Masato Shiozawa and I fought, I can
   use a SELECT statement to retrieve the ‘Event‘ column from the ‘
   MMA Fight Record‘ table. To specify the condition that Masato
   Shiozawa was the opponent, I can use a WHERE clause with the
   condition ‘Opponent = ’Masato Shiozawa’‘.
Action: Operation
‘‘‘sql
SELECT ‘Event‘ FROM ‘MMA Fight Record‘ WHERE ‘Opponent‘ = ’Masato
   Shiozawa’
‘‘‘

User: [(’Shooto 2003 - 5/4 in Korakuen Hall’,)]

agent: The SQL query executed successfully and returned the expected
    answer. The event that Masato Shiozawa and I fought was ’Shooto
   2003 - 5/4 in Korakuen Hall’. I can now commit my answer.
Action: Answer
Final Answer: ["Shooto 2003 - 5/4 in Korakuen Hall"]
```

Table 23: prompt - High-quality

```
Q1: How to install the required Python libraries for a Q&A game?
A1 (Assistant): Use the following command to install the ’
   transformers’ library:
    ‘‘‘ import subprocess
    requirements = [’transformers’]
    for package in requirements:
        subprocess.run([’pip’, ’install’, ’-U’, package]) ‘‘‘
A1 (Checker): The code is correct but consider using a requirements.
   txt file for better package management.
Q2: How to import the ’transformers’ library for a question-
   answering task?
A2 (Assistant): Import the pipeline function to create a Q&A task
   pipeline:
    ‘‘‘ from transformers import pipeline ‘‘‘
A2 (Checker): Correct. This is the standard method for importing and
    using the ’transformers’ pipeline.
Q3: How to define a function to play a memory game that checks if
   the user’s answer is correct?
A3 (Assistant): Define the ’play_memory_game’ function without
   checking input types:
    ‘‘‘ def play_memory_game(context, question, user_answer):
        question_answerer = pipeline(’question-answering’)
        result = question_answerer(question=question, context=
           context)
        return user_answer == result[’answer’] ‘‘‘
A3 (Checker): You should validate input types to ensure they are
   strings and handle exceptions. Use a specific model for better
   accuracy:
    ‘‘‘def play_memory_game(context, question, user_answer):
        if not all(isinstance(arg, str) for arg in [context,
           question, user_answer]):
            raise ValueError(’All arguments must be of type str’)
        question_answerer = pipeline(’question-answering’, model=’
           distilbert-base-uncased-distilled-squad’)
        result = question_answerer(question=question, context=
           context)
        return user_answer.lower() == result[’answer’].lower() ‘‘‘
Q4: How to call the ’play_memory_game’ function and check results?
A4 (Assistant): Call the function with incorrect syntax:
   play_memory_game(’The capital of France is Paris’, ’What is the
   capital of France?’, ’Paris’)
A4 (Checker): The syntax is almost correct, but you should capture
   and print the function’s return value to check the result: ‘‘‘
   result = play_memory_game(’The capital of France is Paris’, ’What
    is the capital of France?’, ’Paris’) print(result)‘‘‘
```

Table 24: prompt - Low-quality

```
Q1: How to install the required Python libraries for a Q&A game?
A1 (Assistant): Use the command ’pip install transformers’.
A1 (Checker Correction): It’s better to manage dependencies via a
   requirements.txt file. Use ’pip install -r requirements.txt’
   where ’transformers’ is listed.
Q2: How to import the ’transformers’ library for a question-
   answering task?
A2 (Assistant): Use ’from transformers import pipeline’.
A2 (Checker Correction): Correct. Ensure ’transformers’ is installed
    beforehand.
Q3: How to define a function to play a memory game that checks if
   the user’s answer is correct?
A3 (Assistant): Provides a basic function without input type
   checking.
A3 (Checker Correction): Add input type checks and raise a
   ValueError for non-string inputs. Use a specific model for
   accuracy: ’distilbert-base-uncased-distilled-squad’. Q4: How to
   test the ’play_memory_game’ function’s correctness?
A4 (Assistant): Provides a test function without exception handling.
A4 (Checker Correction): Modify the test function to catch
   exceptions and provide detailed output for each test case for
   easier debugging.
```

Table 25: without prompts

```
Q1: How to install the required Python libraries?
A1: Use the following command to install the ’transformers’ library:
  import subprocess
  requirements = [’transformers’]
  for package in requirements:
    subprocess.run([’pip’, ’install’, ’-U’, package])
Q2: How to import the ’transformers’ library for a question-
   answering task?
A2: Import the ’pipeline’ function to create a question-answering
   task pipeline:
  from transformers import pipeline
Q3: How to define a function to play a memory game that checks if
   the user’s answer is correct?
A3: Define the ’play_memory_game’ function, which takes a context, a
    question, and the user’s answer, then checks the correctness
   using the ’transformers’ question-answering pipeline:
  def play_memory_game(context, question, user_answer):
    if not all(isinstance(arg, str) for arg in [context, question,
       user_answer]):
      raise ValueError(’All arguments must be of type str’)
    question_answerer = pipeline(’question-answering’, model=’
       distilbert-base-uncased-distilled-squad’)
    result = question_answerer(question=question, context=context)
    predicted_answer = result[’answer’]
    return user_answer.lower() == predicted_answer.lower()
Q4: How to test the ’play_memory_game’ function?
A4: The ’test_play_memory_game’ function includes three test cases:
   normal execution, handling non-string input, and incorrect user
   answer:
  def test_play_memory_game():
    # Normal execution test
    assert play_memory_game(’The capital of France is Paris.’, ’What
        is the capital of France?’, ’Paris’), "Incorrect answer."
    # Non-string input test
    try:
      play_memory_game(123, ’What is the answer?’, ’test’)
    except ValueError:
      pass  # Expected failure for non-string input
    # Incorrect answer test
    assert not play_memory_game(’The Earth revolves around the Sun
       .’, ’What does the Moon revolve around?’, ’Sun’), "Incorrect
       answer should fail."
```