TA-Agent: Tool Augmented Based on Iterative Ideas in Software Engineering

Tianming Ma[1,a], Wei Han[2,b], Yuzhi Mu[2,c], Zihao Wang[2,d], Xuefan Xu[2,e], Junyu Shen[1,f], Jiansheng Wang[1,g], Jian Zhang[2]*

[1]The College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai, China

[2]DataGrand Inc., Shanghai, China

[a]M.Taurus9527@gmail.com, [b]hanwei@datagrand.com, [c]muyuzhi@datagrand.com, [d]wangzihao@datagrand.com, [e]xuefan_xu@outlook.com, [f]1000528078@smail.shnu.edu.cn, [g]1000528087@smail.shnu.edu.cn, *Corresponding author: zhangjian@datagrand.com

***Abstract*—Humans have long been pursuing Artificial Intelligence (AI) that equals or surpasses the human level, and AI agents are considered a promising vehicle for achieving this goal. Artificially intelligent agents are artificial entities that can perceive the environment, make decisions, and take actions. Agents can solve practical tasks based on existing tools. However, the current tools-based Agents suffer from the drawbacks of incomplete loading of tools libraries and redundancy in tools creation. To address these two issues, we introduce the Tools Augment Agent (TA-Agent) based on the iterative thinking of software engineering. Our automation paradigm uses LLM-based agents to achieve enhancement automation through reasoning and acting recall tools. It is designed to enhance existing tools according to human task requirements without creating new tools as much as possible and to continuously optimize and improve the quality of existing tools through iterations. Finally, the tools-augmented task construction and execution processes are explained in detail through experiments, proving the feasibility of TA-Agent and revealing the possibility of a new paradigm of Agent-driven automation. The code is available at https://github.com/Taurus9527/Tool_Augment.**

Keywords-LLM-based agents; Software Engineering and Iteration; ReAct; Retrieval-Augmented Generation; Tools augment

I. INTRODUCTION

Recent research has shown that large language models(LLMs)[1] can solve natural language tasks excellently. From the early days when LLMs could only answer simple tasks, they can now create tools to solve problems. The capabilities of LLMs are advancing rapidly, and the tasks they can accomplish are becoming increasingly complex. LLM-based agents[1] aim to use the ability of LLMs to understand and generate text, so that LLMs behave more intelligently and naturally when interacting with users or other systems. As agents technology becomes increasingly widely used, agents continuously improve their task-completion abilities. Although agents have made great progress in handling intricate tasks, there are hidden problems in the tools manufacturing[2] process. Continuously manufacturing new tools will cause a large redundancy in the Tools Set. Retrieval of related tools becomes more difficult, resulting in reduced agents efficiency.

Based on the concept of Software Engineering Iteration[3],we introduce a Tools-Augmented agent (TA-Agent) that incorporates unit tests[4] into its process. This agent[5] is designed to enable LLMs to evaluate the effectiveness of tools in relation to specific functions and tasks during tools recall[6], rather than resorting to creating new tools when existing ones fail to address the task. TA-Agent solves two problems when agents use tools: 1) Too many tools can be fully loaded simultaneously. 2) The tools produced independently by LLMs in the Tools Set are of low quality and have a single ability.

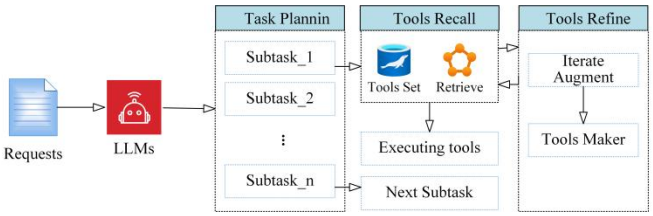

Figure 1 TA-Agent, Task Planning: Splitting requests into subtasks; Tools Recall: Recall tools based on subtasks; Tools Refine: When the tool doesn't fit perfectly, choose to enhance the tool.

To explore the feasibility of TA-Agent, we instantiate TA-Agent, which integrates tools recall and tools augment in a unified framework in Figure 1. For the framework's architecture, we based its design entirely on JSON structure and Python code[7], leveraging LLMs' pre-trained[8] coding corpora to facilitate agents' comprehension and execution of tool optimization tasks. Specifically, TA-Agent uses a JSON structure to organize the input and output of each part to standardize the data and uses Python functions as the presentation form of the tools. After receiving specific tasks, the TA-Agent can generate corresponding task plans[9]. TA-Agent needs to focus on one point for the operation: TA-Agent has a rational plan for demand.

To experimentally verify our method, we conducted verification experiments to demonstrate that TA-Agent can optimize tools based on actual tasks and enhance tools through iterative ideas. Our contributions are listed as follows:

- We propose TA-Agent, an automated paradigm for tools augment that integrates agents further to achieve increasingly higher quality tools in the Tools Set.
- We instantiated TA-Agent, which is designed to augment the tools. We use the retrieval-augmented generation method and the idea of unit testing in software engineering to recall tools and enhance them.
- We verify the feasibility of TA-Agent through experimental analysis and explore enhancement opportunities across various fields, combining the resulting insights.

II. RELATED WORK

A. LLMs-based agents

As LLMs continue to showcase remarkable capabilities and gain widespread popularity, researchers are increasingly integrating these models into the core architecture of AI[10] agents. They utilize LLMs as the primary brain or controller of these agents, while also expanding their perception and action capabilities through techniques such as multi-modal perception and tools utilization[5].

In *Tool Makers[2],* introduce the concept that LLMs can generate tools based on examples, which can then be stored in a Tools Set for use by other LLMs. By employing LLMs with small parameters or low resource consumption, these agents can create reusable tools[2] to address new tasks effectively. This approach not only enhances the versatility of LLM-based agents but also enables them to adapt to a wide range of scenarios by leveraging existing knowledge and learned tools.

B. Reasoning + Acting

Reasoning + Acting(ReAct)[11] is a general paradigm that combines reasoning and behavior with LLMs. It generally consists of three parts: Thought, Action, and Observation. Some specific tasks may be slightly adjusted. This paradigm is used to solve various language reasoning and decision-making and has excellent results. ReAct prompts LLMs to generate verbal reasoning traces and task-relevant actions in an interleaved manner, which allows the model to perform dynamic reasoning to create, maintain, and adjust high-level action plans while also interacting with the external environment (such as Wikipedia) to incorporate additional information to reasoning. In the paper *A Dataset for LLM Question Answering with External Tools[12]*, the author used the ToolQA data set to evaluate the ability of ReAct. The results showed that the LLaMA-2 13B model based on ReAct performed 6.2% higher than the LLaMA-2 13B model on difficult tasks. For simple tasks, the score is as high as 29 %.

Integrating the ReAct paradigm into each stage of the agents allows for the full utilization of LLMs' advantages in discovering logical relationships and reasoning capabilities in language. This integration enhances LLMs' proficiency in handling complex problems. ReAct will expand LLMs' knowledge base and increase LLMs' external capabilities. The role of the ReAct method is to coordinate the acquisition of LLMs and external information, and interact with other tools.

C. Software Engineering and Iteration

Software Engineering[3] is applying systematic, standardized, and measurable methods to develop, operate, and maintain software. Iteration in the Software Process, Software Engineering is regarded as a collection of related activities that constitute the entire process of producing software systems. This process can be viewed as a whole, understood, and analyzed through reasoning. However, software development typically does not progress directly from application concepts to operating systems linearly. In this process, there will be various deviations from linear progress, which can be called "iteration", "repetition", "backtracking" or "redoing". In the Iteration of the software development process, these activities are collectively called "Iteration". Iteration is key to managing changes in the software development process, which may involve modifications to previous decisions and actions and modifications to the resulting object. Iteration is not only necessary to correct errors, but also a natural part of the refinement and evolution of software systems. This need for change doesn't just stem from previous mistakes, but is an inevitable part of the software development process.

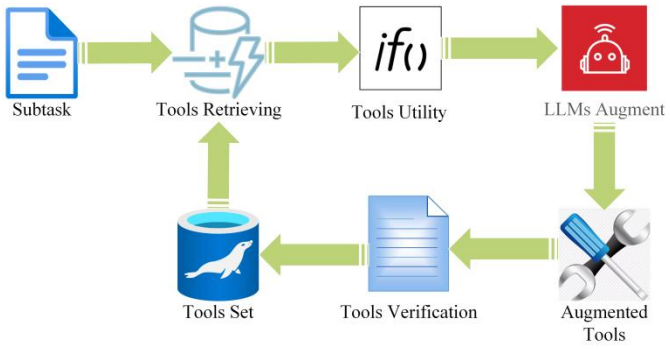

Figure 2 TA-Agent: determine whether a tool can be optimized; and validate the optimized tool.

D. Retrieval-Augmented Generation

The Retrieval-Augmented Generation(RAG)[13] method aims to strengthen the capabilities of LLMs. Its core is to integrate external data resources to expand the existing knowledge base of LLMs. In addition, data resources may include external vector databases, external knowledge graphs, or even directly integrate existing ElasticSearch[14] or search engines of production environments. Although access methods vary, they all differ in subtle ways. The first task of this method is to retrieve user context-related information from external data sources, then append it with the existing knowledge of provide an embedded of the integrated information for use by LLMs. Ultimately, LLMs generate responses based on their intrinsic cognitive properties.

III. METHODOLOGY

The agents' ability to handle complex tasks needs to be improved. We designed TA-Agent using the ReAct method and LLMs with better effects to achieve the decomposition, solution process, and result output of intricate tasks. Figure 2 shows that according to the subtasks retrieval, enhancement, and validation tools. We use an RAG-based approach to retrieve appropriate tools from the Tools Set. For tools that cannot be fully adapted, TA-Agent will not create new tools. Instead, it will use the idea of iterative optimization based on the existing tasks and the existing tools' capabilities to achieve what can enhance the capabilities of the current tool.

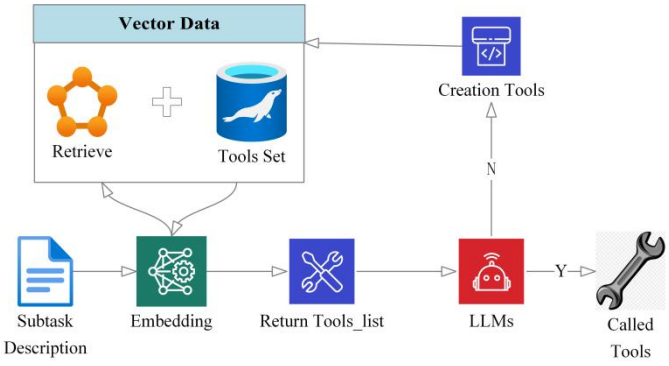

Figure 3 Recall tools.

A. *RAG-Based Retrieval Tools*

Task Planning The synergy between ReAct and agents enhances the capabilities of LLMs, making them less prone to problems such as hallucinations and error propagation during the reasoning process. This makes LLMs even better at handling complex problems. We integrate ReAct into TA-Agent. When TA-Agent faces complex tasks, it provides the solution steps, solution actions, and action results of the task, allowing LLMs to imitate and learn. It also has the ability to help LLMs obtain information, output content, and execute decisions. For specified task requirements, the ReAct command will automatically complete the knowledge and relevant information that LLMs should have, and then let the LLMs make decisions[15].

Tools Retrieval TA-Agent understands and executes the steps planned by ReAct. The LLMs' capabilities can solve some of the subtasks, while other tasks require the use of tools. In Figure 3, we vectorize the Tools Set and store the vector in the database. The subtask is vectorized when performing tools retrieval and the Top-k list results are retrieved. The tools in the list are matched with the subtasks to verify whether these tools can suit the subtasks. LLMs complete this process. When the above k tools are not adapted, we turn to the Tools Maker[2], extract key elements from the information of the current subtask, and then use LLMs to generate a new tool to solve the current subtask. The new tool produced is described and documented, then incorporated into a Tools Set for future use.

B. *Iteration-Based Tools Augmented*

Tools Augment In the tools augment phase, we are faced with the fact that the current tools cannot fully meet the needs of the current subtask. This may be because the capabilities of the current tools are not powerful enough or cannot be perfectly adapted. For example, the current tools may be designed to read documents in “txt” file format, but the task requires reading a “xlsx” table. For this situation, we iteratively refine the existing tools for enhancement and optimization and conduct automated testing[16] of the enhanced tools. Optimization aims to make the tool meet the current subtask requirements while retaining the tools' original functionality. Automated testing is a test of the enhancement process using automated scripts to verify the quality and stability of the entire enhancement process. After judging that the tool is optimizable for the subtask, LLMs will extract the task requirements and compiler feedback to enhance the tool. As in Figure 4, the TA-Agent will refine the tool until it meets the requirements of the test set. This enhancement process requires LLMs to carefully analyze the structure and functionality of the current tool and identify areas that need improvement. Subsequently, we will make corresponding adjustments and improvements to the tool based on the specific requirements of the subtask and the errors reported by the compiler. At the same time, during the optimization process, we will also focus on retaining the core functions and features of the original tool to ensure that the changes will not introduce new problems or destroy the original functions.

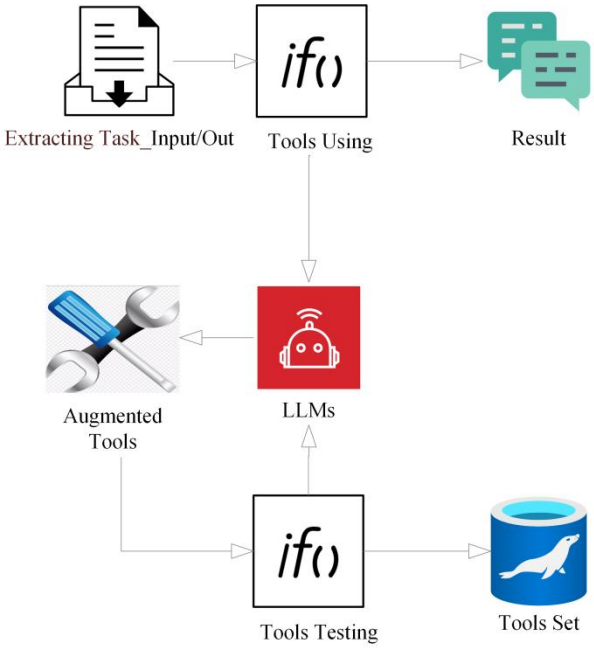

Figure 4 Tools augment and tools validation. When the tool is not perfectly adapted to the task at hand, LLMs make an iterative enhancement to the tool, considering the task requirements and Python feedback.

Tool Verification When optimizing a tool, we want to ensure that the changes do not destroy the functionality of the original tool. To verify this, we used unit testing. Figure 4 shows that we extracted relevant examples from the description of the original tool, and then used these examples and the current subtask to form a test set to verify the new tool. Through unit testing of the new tool, verify whether the new tools' performance in solving the sample problem is feasible.

Through validation, the tool will become more powerful and able to solve the current subtask while retaining all the functional features of the original tool. This optimization process transcends mere enhancements, delving deeply into the extraction of current subtasks and tools' descriptions. The new tool will also generate a new description to summarize the tools' functional description, performance improvements, and more detailed usage examples, providing users with more comprehensive and clear guidance and application directions.

IV. PROOF-OF-CONCEPT EXPERIMENT

To validate the feasibility of tools recall and enhancement, we built our agent based on the ReAct method, the langchain[17] framework, and the GPT-4[18]. By building the ReAct prompt[19], you can task planning and use Langchain to build RAG question and answer instances and complete tools retrieval work. We aim to optimize the quality of our tools to be more efficient and applicable to a broader range of applications.

ReAct Prompt

Question: Write an announcement about housing in the apartment "Skylit Midtown Castle" and specify how much it costs for a month's stay.
Thought 1: I need to break down the question into sub-tasks.
Action 1: Decompose[question]
Observation 1: \nTask_1:I need to inquire about the apartment "Skylit Midtown Castle".\nTask_2:... \nTask_3:...
Thought 2: I need to fix Task_1 first.Read the table in path "/TA-Agent/data/airbnb/Airbnb_Open_Data.xlsx" to get the information of "Skylit Midtown Castle"...
Action 2: Get[Tool] Or Maker[Tool]
Observation 2: ans_Tool_1
...
Thought 6: Use this function to calculate one month's rent for the apartment "Skylit Midtown Castle"...
Action 6: Calculator[Money] Or Refine[Tool]
Observation 6: ans_Money
Thought 7: Write a notice...
Action 7: Write[Notice]
Observation 7: ans_Notice
Thought 8: After performing the above steps, the task is completed
Action 8: Finish[Task]

Figure 5 Thought example of ReAct.

A. Generate ReAct Prompt

We give a typical daily life scenario. The rental agency needs to extract relevant information about the "Skylit Midtown Castle" house based on the house data, generate a housing announcement, and give the rent information for this house.

Figure 5 shows that the TA-Agent has made a reasonable plan for the task. The housing data is stored in "xlsx" format, which contains various information about the housing. LLMs need to read the housing information, retrieve the relevant "Skylit Midtown Castle" information, and return it to GPT-4. GPT-4 generates housing announcements and rent information based on reference information. If there is a service fee, additional service fees will be added. This exemplifies the process of demonstrating enhancements to our TA-Agent tool by relying on actual tasks:

- As shown in Figure 5, according to question Task_1, use the tools to query the hotel information under the path "/TA-Agent/data/airbnb/Airbnb_Open_Data.xlsx" and accurately extract the information of the "Skylit Midtown Castle" hotel;
- Calculate the rent information based on the hotel information of "Skylit Midtown Castle" and accurately return the tools calculation results to GPT-4, which requires dynamic and flexible extraction of calculation information;
- GPT-4 generates a rent announcement based on hotel information and rent results.

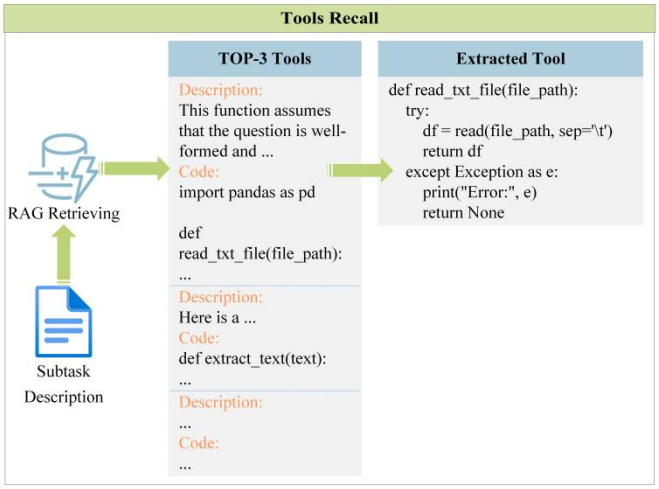

Figure 6 Tools recalls' result.

B. Tools Recall

The TA-Agent uses the Langchain framework to build a dialogue system. The tools are stored in JSON format and the M3E-Base pre-trained model is used to vectorize the tools in the Tools Set for retrieval. TA-Agent needs to retrieve the tool according to the description of Thought 2. RAG will return the Top-3 results and send them to GPT-4 to determine whether they are available or optimized. If the GPT-4 judgment fails, try to make a tool to solve the current problem, record its description, and store it in the database.

Figure 6 shows the tool results of M3E-Base recall based on subtask description, including some description statements and Python code. After TA-Agent gets the returned results, it needs to process the results and extract its usage examples so that TA-Agent can apply them to the current task based on the examples.

C. Tools Augment

LLMs determine whether the tool can be used or optimized, use the Python interpreter to call the tools, and determine whether the task is completed based on the feedback results. As shown in Figure 7, when the feedback results do not reflect the completion of the task, LLMs need to enhance the tool based on the task description, task goals, and compiler feedback results. At the same time, three usage examples of the tool and current task requirements were extracted to form a test set of four examples to verify the enhanced tool. Verified the tool not only enhances the tool's ability to solve current tasks but also retains its original capabilities. The function calling method was optimized before and after the *read_txt_file()* was enhanced, and the tools' ability to solve multiple expressions was increased. We generate new relevant information by extracting descriptions of new tools and test set examples, and save the information and tool pairs to the toolset in a dictionary format.

D. Result Explanation

As shown in Figure 7, the Python interpreter executes the *read_txt_file()* with an error. GPT-4 optimizes the tool with interpreter error, task requirements, and the prompt. The goal is

to solve the error and retain the original function. We add the inputs and outputs of the current task to the test set to validate the enhanced tools. We used the new test set to test if the tool resolved the error. In Figure 7, the tool can pass the validation of the test set, showing that the tool can read 'txt' and 'xlsx' files.

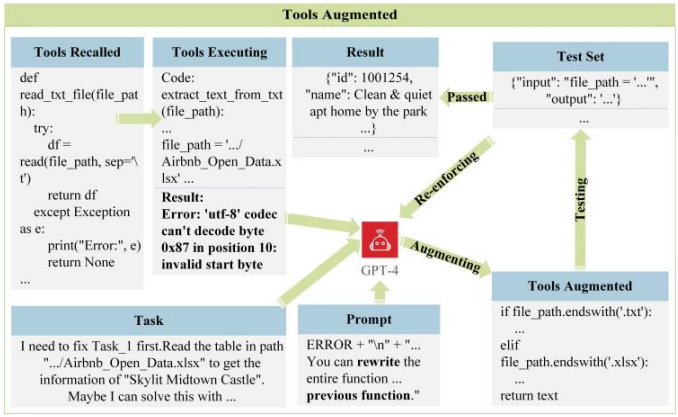

Figure 7 Tools augmented.

V. CONCLUSION

We propose tools augment automation paradigm based on software engineering iterations for tools recall and enhancement, integrating them into task planning and tools utility by leveraging LLM-based agents. This solves the "Tools Explosion" problem when LLMs manufacture their own tools. Through the instantiation of TA-Agent, we illustrate how LLM-based agents can feasibly manage the decision-making process of complex tasks. Our concept proves the "Tools Explosion" problem that arises in the case of LLMs manufacturing tools and solves the shortcomings of incomplete loading of tools libraries, redundancy in tools creation, and waste of space storage resources. Our findings contribute to the growing body of research in intelligent automation and highlight the important role that LLM-based agents can play in improving efficiency and flexibility across industries. As the adoption of automation technology continues to expand, we expect TA-Agent to serve as a catalyst for further developments in the automation space, leading to greater efficiency, less human intervention, ultimately, a more streamlined and intelligent workflow ecosystem.

REFERENCES

[1] L. Wang et al., "A survey on large language model based autonomous agents," Frontiers of Computer Science, vol. 18, no. 6, pp. 1-26, 2024.

[2] T. Cai, X. Wang, T. Ma, X. Chen, and D. Zhou, "Large language models as tool makers," arXiv preprint arXiv:2305.17126, 2023.

[3] M. Dowson, "Iteration in the software process; review of the 3rd International Software Process Workshop," in Proceedings of the 9th International Conference on Software Engineering, 1987, pp. 36-41.

[4] H. Zhu, P. A. Hall, and J. H. May, "Software unit test coverage and adequacy," Acm computing surveys (csur), vol. 29, no. 4, pp. 366-427, 1997.

[5] Y. Qin et al., "Toolllm: Facilitating large language models to master 16000+ real-world apis," arXiv preprint arXiv:2307.16789, 2023.

[6] S. Huang et al., "Planning, Creation, Usage: Benchmarking LLMs for Comprehensive Tool Utilization in Real-World Complex Scenarios," arXiv preprint arXiv:2401.17167, 2024.

[7] Y. Ye et al., "Proagent: From robotic process automation to agentic process automation," arXiv preprint arXiv:2311.10751, 2023.

[8] X. Han et al., "Pre-trained models: Past, present and future," AI Open, vol. 2, pp. 225-250, 2021.

[9] J. Ruan et al., "Tptu: Task planning and tool usage of large language model-based ai agents," arXiv preprint arXiv:2308.03427, 2023.

[10] P. Zhao, Z. Jin, and N. Cheng, "An in-depth survey of large language model-based artificial intelligence agents," arXiv preprint arXiv:2309.14365, 2023.

[11] S. Yao et al., "React: Synergizing reasoning and acting in language models," arXiv preprint arXiv:2210.03629, 2022.

[12] Y. Zhuang, Y. Yu, K. Wang, H. Sun, and C. Zhang, "Toolqa: A dataset for llm question answering with external tools," Advances in Neural Information Processing Systems, vol. 36, 2024.

[13] Y. Lyu et al., "CRUD-RAG: A comprehensive chinese benchmark for retrieval-augmented generation of large language models," arXiv preprint arXiv:2401.17043, 2024.

[14] O. Kononenko, O. Baysal, R. Holmes, and M. W. Godfrey, "Mining modern repositories with elasticsearch," in Proceedings of the 11th working conference on mining software repositories, 2014, pp. 328-331.

[15] H. Yang, S. Yue, and Y. He, "Auto-gpt for online decision making: Benchmarks and additional opinions," arXiv preprint arXiv:2306.02224, 2023.

[16] B. Meyer, I. Ciupa, A. Leitner, and L. L. Liu, "Automatic testing of object-oriented software," in SOFSEM 2007: Theory and Practice of Computer Science: 33rd Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 20-26, 2007. Proceedings 33, 2007: Springer, pp. 114-129.

[17] O. Topsakal and T. C. Akinci, "Creating large language model applications utilizing langchain: A primer on developing llm apps fast," in International Conference on Applied Engineering and Natural Sciences, 2023, vol. 1, no. 1, pp. 1050-1056.

[18] J. Achiam et al., "Gpt-4 technical report," arXiv preprint arXiv:2303.08774, 2023.

[19] Y. Cheng, J. Chen, Q. Huang, Z. Xing, X. Xu, and Q. Lu, "Prompt sapper: a LLM-empowered production tool for building AI chains," ACM Transactions on Software Engineering and Methodology, 2023.