Self-evaluation of Generation Augmented by Tools

DOUGLAS COIMBRA DE ANDRADE[*], Instituto SENAI de Inovação em Sistemas Embarcados, Brazil

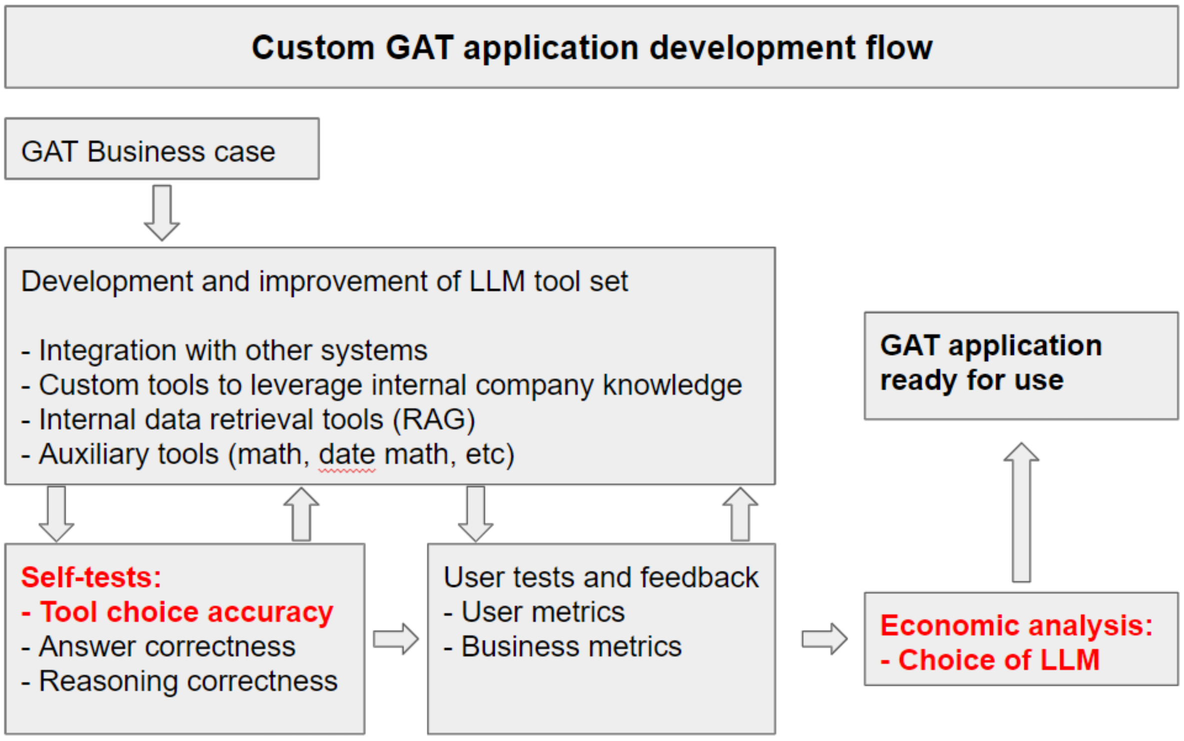
Custom GAT application development flow
GAT Business case
Development and improvement of LLM tool set
- Integration with other systems
- Custom tools to leverage internal company knowledge
- Internal data retrieval tools (RAG)
- Auxiliary tools (math, date math, etc)
Self-tests:
- Tool choice accuracy
- Answer correctness
- Reasoning correctness
User tests and feedback
- User metrics
- Business metrics
Economic analysis:
- Choice of LLM
GAT application ready for use

Fig. 1. Development flow of a GAT application

This paper introduces a novel method for evaluating Generation Augmented by Tools (GAT) systems in industrial settings where custom tools are developed. We propose a self-testing framework that uses high-performing Large Language Models (LLMs) to automatically generate test cases for tool selection and usage. Our method addresses the challenge of assessing GAT applications with proprietary or frequently updated tools, which are not covered by existing public benchmarks. We implement specific tools, including symbolic math solvers, file manipulation utilities, and visualization generators, to

[*] *

Author's address: Douglas Coimbra de Andrade, douglas.coimbra@sc.senai.br, Instituto SENAI de Inovação em Sistemas Embarcados, Avenida Luiz Boiteux Piazza, 574 - Cond. Sapiens Parque - Canasvieiras, Florianópolis, Santa Catarina, Brazil, 88054-700.

ACM XXXX-XXXX/2024/0-ART0
https://doi.org/XXXXXXX.XXXXXXX

demonstrate the framework's versatility. Using our approach, we evaluate multiple LLMs hosted by multiple providers, including both open-source and commercial models, on a set of self-generated test cases. Our results show that native tool-calling capabilities outperform instruction-based approaches, with top models achieving 80% accuracy in tool selection. We also find that smaller, cost-effective models like Claude 3 Haiku and GPT-4o mini can achieve comparable performance to larger models in specific GAT tasks. This work provides a practical methodology for companies to select appropriate LLMs for their GAT applications, balancing performance, cost, and specific task requirements. Source code is available in https://github.com/douglas125/SelfTestingGAT_LLM.

CCS Concepts: • **Computing methodologies → Natural language processing**; *Natural language generation*; • **Software and its engineering → Software verification and validation**.

Additional Key Words and Phrases: LLM, RAG, Tool Calling, Neural Networks, Software Testing

ACM Reference Format:
Douglas Coimbra de Andrade. 2024. Self-evaluation of Generation Augmented by Tools. 0, 0, Article 0 (2024), 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Large language models (LLMs) revolutionized multiple tasks such as text summarization, language translation, question answering, and code generation, enabling state-of-the-art performance in these tasks.

However, keeping LLMs up to date with current or proprietary, non-public information involves either a process of continuous fine tuning, which tends to be be expensive and probably not feasible, ways to modify the prompt instructions to include relevant data in a structured form or tools that enable on-demand retrieval.

The approach of Retrieval Augmented Generation (RAG) or, more generally, Generation Augmented by Tools (GAT) enables including relevant, up-to-date information to be analyzed by the model at the cost of increased number of tokens. The distinction between RAG and GAT is relevant because tool use (also named function calling) enables LLMs to perform operations that they are unable to do natively without necessarily needing to retrieve new data. Some examples are: complex mathematical operations, generation of graphs and charts, image generation, text-to-speech conversion, and sentiment analysis.

The field of LLMs is very active at this time. Multiple private and open LLM choices exist, with private LLMs outclassing open source ones [10]. Notably, the best performers in the field at the moment of this research are Claude 3.5 and GPT 4o (developed by Anthropic and OpenAI, respectively).

Another concern is about data privacy. Since RAG approaches may involve retrieval and use of internal information, multiple companies still refuse to use the services of LLM start-ups, with their services often blocked in corporate PCs. Thus, trusted clouds (like AWS, Azure and GCP) tend to be the solution of choice when adopted.

The rapid development of LLMs and their integration with tools has created a need for robust evaluation methods, especially in industrial settings where specific, proprietary tools may be used. This paper addresses this gap by proposing a novel self-evaluation method for GAT systems. Figure 1 describes the main steps to develop a custom GAT application in an industry and highlights (in red font) the steps where this research fits. Typically, a business case guides the development. Tools need to be created to integrate the LLM wih internal systems, data and specialized knowledge. Our GAT self-testing framework helps to evaluate the GAT tool selection capabilities and pick the most adequate LLM for the specific case.

Common problems while developing GAT-LLM applications within the specific context of a company are:

- How to select the cheapest LLM that still provides acceptable performance?
- How to improve prompts and tool descriptions to increase the accuracy when selecting a specific tool?
- Can we ensure that improving the ability of calling a tool / introducing a new one will not decrease the ability to select the other tools?

To address these problems, a new evaluation method for GAT is presented. We propose a novel way of using a LLM that performs best in open benchmarks (thus also potentially more expensive) to automatically generate tool use test cases. Those may be combined with manually generated tests to select an adequate LLM for production use, taking into account price, speed in addition to test performance.

We also implement multiple tools whose combined use can perform various real-life tasks, such as video editing from subtitles (using ffmpeg and reading subtitles), repository documentation (reading source code in a folder and writing files to that folder), visual summarization of webpages (retrieve webpage contents and use plot / graph visualization tools) and others.

The contributions of this paper are the following:

- A minimalistic open source framework for function calling compatible with multiple LLMs and providers.
- An open-source repository with the implementation of tools spanning a wide range of meaningful uses.
- An automated method to generate relevant test cases (GAT self testing) that enables evaluation of specific GAT-LLM systems and verifies the correctness of tool selection.
- A proposal about how to use GAT self testing to choose an appropriate LLM and/or evaluate prompt and tool description changes.
- Detailed benchmarks comparing metrics of publicly available LLMs when calling the tools implemented.
- Source code to use GAT with the proposed tools and replicate all tests in https://github.com/douglas125/SelfTestingGAT_LLM.

This work is organized as follows: Section 2 describes how GAT-LLM systems are built and previous work on evaluating GAT and LLMs.

Section 3 presents the methodology, including the design of prompts for self-test generation and the metrics used for evaluation. Section 4 details the results of the experiments, including a comparison of different LLMs' performance on the self-generated tests. Finally, Section 5 concludes the paper and discusses future research directions.

2 BACKGROUND

2.1 GAT-LLM architecture

The introduction of tool use in LLMs opens multiple possibilities, such as:

(1) Use of up-to-date information
(2) Use of custom, internal data from private databases (RAG)
(3) Use of specialized tools (for example: read and write files, access the internet, solve equations)

GAT-LLM systems need the following components:

(1) A base LLM that will "understand" the instructions, the description of the tools and generate a structured sequence of tokens that can be parsed into a tool use call
(2) A detailed description of each tool, including their arguments, use cases, cases when not to use and response
(3) The implementation of each tool using "conventional" programming languages (Python, C++, etc)

Mathematically, the goal is to maximize the probability that the LLM will output a sequence of tokens that can be parsed as a tool use request. This requires including in the LLM prompts detailed specifications of what sequences of tokens have to be generated when external data is required, and how the answer is going to be provided. In addition to that, it is necessary to use the function call end string as a new stop_sequence for the LLM, which often requires using the LLM in streaming mode if new stop_sequences are not supported (otherwise, the LLM will generate invented values for the results of the tools).

GAT augments LLMs in so many ways that most commercial implementations now provide a native form of function calling ([3], [11], [5], [1]). Since not all available LLMs have function calling built-in, we find it fair to also evaluate whether using native function calling capabilities is important when developing a GAT system.

2.2 Previous Work

LLMs have sparked great interest in the research community. This growing interest lead to the development of multiple evaluation benchmarks, like the MATH dataset [6], the Graduate Level Reasoning Benchmark [14], the HumanEval Code Benchmark [12] and many others. All of these are often used when new LLMs are released to provide an idea about their reasoning capabilities.

Most of the benchmarks, however, do not evaluate the ability to invoke tools correctly. For this purpose, the Berkeley Function Calling Leaderboard was created, comprising multiple tasks including scripting, Python, SQL and the ability to perform parallel calls [19]. This benchmark helps understand whether LLMs are capable of selecting the correct tools to perform a fixed number of tasks.

Ensuring good performance of LLMs in industrial contexts is a relevant topic with plenty of active research. For example, the ability to generate structured outputs is needed to integrate LLM with legacy systems, and evaluating this capability was investigated in [8]. Document question answering is also relevant for industrial use since documentation may rapidly change in many cases, and yes-no questions seem to be a better evaluation choice for the task (instead of multiple choice questions) [13]. Industry-specific knowledge assessment has been investigated, for example, in the cases of mastery of knowledge and skills in the heating, ventilation and air conditioning (HVAC) industry [9] and understanding customer satisfaction in e-commerce [15], among others.

The problem of customizing tests using self-supervision to ensure good performance in industrial LLM applications is discussed by [7]. The approach of using data generated by a LLM with performance known to be good as benchmark for others has similarity with ours. However, the authors focus on directly monitoring the LLM behavior (without focus on tool use).

The problem that still remains is the one of testing a GAT system developed with specific tools within the context of a company. Fixed public benchmarks need to evolve to accommodate the increased capability of larger models, which is why they need to become more challenging.

Companies implementing GAT systems often face budget constraints and performance requirements that necessitate a balance between cost and capability. While the highest-performing LLMs may offer superior results, they can be prohibitively expensive for many applications. By developing specific evaluation tools, companies can identify LLMs that meet their particular needs at a lower cost. This approach allows for the selection of models that perform adequately on task-specific metrics while optimizing for factors such as inference speed and API costs, which are crucial in real-world deployments.

A recent, survey on evaluating LLMs provided a comprehensive analysis of multiple tasks and benchmarks used to evaluate LLMs [4]. This survey highlights the importance of continuing to improve and, more importantly, making benchmarks specific to the task that needs to be achieved. In particular, the suggested research topic of "Dynamic and Evolving Evaluation" is the main topic of this work, with focus on verifying LLMs capability of making the correct tool choices in a scenario where tools are not available beforehand. Instead, they are implemented within a company to perform specific GAT tasks.

While these existing benchmarks and evaluation methods provide valuable insights into LLM performance, they do not address the specific needs of companies implementing custom GAT systems. Our proposed self-evaluation method bridges this gap by allowing for the assessment of LLMs on company-specific tools and tasks. Although extensive approaches to LLM testing have been discussed recently, to the best of our knowledge, no self-assessment method (like ours) has been proposed to address the problem of testing internal GAT applications.

3 METHODOLOGY

3.1 LLM prompting

Though multiple LLMs exist and their task is the one of predicting the next token, their implementations vary considerably. For example, Llama 2 [16] uses the following format:

```
<s>[INST] <<SYS>>
{{ system_prompt }}
<</SYS>>

{{ user_message_1 }} [/INST] {{ model_answer_1 }} </s>
<s>[INST] {{ user_message_2 }} [/INST]
```

Llama 3, on the other hand, introduces some changes to the chat prompting format:

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>
You are a helpful assistant<|eot_id|><|start_header_id|>user<|end_header_id|>
What is the capital for France?<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

The input to most commercial LLMs is in JSON format, as follows (example from [2]):

```
{
    "system"="You are a world-class poet. Respond only with short poems.",
    "messages"=[
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": "Why is the ocean salty?"
                }
            ]
        }
    ]
}
```

Where the full chat history has to be sent to the LLM provider, modifying "user", "assistant", "system" and other role keywords. How those LLMs parse the JSON into proper prompts is not disclosed.

Because of this high variability in how system messages and user/bot interactions are parsed into proper LLM-specific prompts, we evaluate native tool calling capabilities (when available) along with instruction-based tool calling.

3.2 GAT instructions

Enabling LLMs to use tools requires providing a detailed description of the tool, its arguments, use cases and cases where the tool should not be used.

When tools have distinct functionalities, the LLMs can easily differentiate between them. However, for tools with overlapping capabilities, precise instructions are crucial. For example, when deciding between a tool that generates QR codes and one that plots graphs, or between a tool that evaluates numerical expressions and one that solves symbolic equations, the LLM needs clear guidelines. An instance of this could be the difference between "'Create a visual representation of the equation y = 2x + 3"' (which might use a plotting tool) and "'Generate a QR code containing the text $y = 2x + 3$ "' (which would use a QR code generator).

3.3 GAT evaluation

Though there are only a handful of LLMs to choose from, the amount of GAT applications that can be built is enormous. Each company may have specific needs for multiple GAT to retrieve proprietary data and perform tasks. This can be achieved by customizing the tools.

While there are multiple data sets and benchmarks for LLM evaluation, it is not possible to build a static benchmark for all GAT applications. To solve this issue, we introduce a method for GAT Self Testing focused on generating test cases for LLMs equipped with a set of tools that is not known beforehand.

In a development environment that may contain multiple tools developed by different teams, the description of new tools may lead the LLM to choose the new tool in situations where the expected behavior was the old one. To make sure that the nuances of different tools are covered, we propose to generate tests for pairs of tools in addition to individual ones. For example: to test the tools solve_with_python and solve_symbolic, whose descriptions may overlap and intertwine if not written properly, one possible test question is "Solve the equation $x^2 - 25x + 156$' and give the solution with the digits reversed'. The symbolic tool should be used to compute the solution and the Python tool is appropriate to reverse the digits.

3.4 Self-test generation prompt design

During the design of the prompt used to generate the self tests, the initial task description was a variation of the following:

"Analyze carefully your tools. Generate a set of NTESTCASES questions that can be answered using exactly those tools. All tools should be necessary to provide the answer."

The string "NTESTCASES" gets replaced by the actual number of desired tests. However, the LLMs tend to oversimplify the task by generating fewer cases and simple cases that require only one tool.

To mitigate this issue and increase the probability of obtaining more complex and meaningful test questions, the following extra instructions were added:

- "The questions should be as different as possible from each other.": This instruction significantly reduced the number of similar questions generated, which would only change a number from one question to another.
- "Never generate questions that would require only one tool to be answered.": Along with specific requests for questions that needed two tools to be answered, this extra sentence improved the probability that the LLMs would generate questions that really needed more than one tool. Interestingly, when it is difficult that a problem would require two specific tools to be solved (e.g. use_ffmpg and make_qr_code), the LLMs still generated only questions that required one of those tools.

3.5 Metrics

We choose to keep two simple metrics for the purpose of evaluation of GAT self-testing: accuracy and score. For each question, if the LLM picks the exact tools needed to solve the task, it is awarded accuracy 1. The LLM's score on each question is computed as the intersection over union of the correct tools and the tools chosen by the LLM. The inclusion of a score metric intends to reward LLMs that were able to at least pick one correct tool among the ones considered necessary to solve the task.

4 RESULTS

4.1 LLMs

In the case of Cohere, we set $force\_single\_step = True$ to ensure that only one function is called at a time, to be able to compare with other models. While using one tool at a time seems to be the recommended approach by commercial LLMs, empirical evidence supporting this in all scenarios is limited. The trade-offs between single and multiple tool calls in various contexts requires further investigation.

4.2 Implemented Tools

To evaluate a wide array of tasks while maintaining a manageable scope, we implemented a set of specific tools. These tools were selected to cover common industrial needs and to allow for complex, multi-step operations. The following list provides a brief description of each tool's functionality:

- **do_date_math**: Performs date calculations by adding or subtracting time intervals from a given date.
- **read_write_user_details**: Reads or writes relevant information about the user being assisted, ensuring data integrity and user confirmation.
- **make_custom_plot**: Generates images or plots using custom Python code with numpy and matplotlib dependencies.
- **solve_symbolic**: Solves symbolic mathematics problems using the SymPy library for complex equations and expressions.
- **solve_numeric**: Evaluates numerical expressions using the NumPy library for direct numeric calculations.
- **get_url_content**: Retrieves and processes content from one or more internet URLs, allowing for specific information extraction.
- **make_qr_code**: Generates QR code images with customizable parameters such as error correction and box size.
- **read_local_files**: Reads the contents of one or more local text files.
- **write_local_files**: Writes content to a local text file.
- **read_file_names_in_local_folder**: Retrieves the list of file names contained in a specified local folder.
- **use_ffmpeg**: Executes ffmpeg commands for video manipulation tasks.
- **plot_with_graphviz**: Creates graph visualizations using custom Python code with the pydot library.

Note that these tools and their combinations enable useful tasks, such as:

- Remember user details and summarize daily useful information from the internet for a user.
- Edit videos with ffmpeg based on natural language if a subtitles file is provided.
- Analyze project files and/or source code and write documentation files.
- Perform mathematical operations to create graph and chart visualizations.

4.3 Self-generated Tests

To self-generate the test cases that were used to evaluate a potential GAT application that uses the tools implemented, the choice was to use known best-in-class LLMs (Claude 3.5 Sonnet and GPT4o at the time of this study).

However, to quantify whether there is any loss in the quality of the tests generated, tests were also conducted using smaller versions of those LLMs. Initially, the LLMs were prompted to generate use cases for the tools individually. However, manual inspection of the questions generated immediately showed that the test cases were too shallow and failed to test complex cases where the tool choice may not be obvious (among two or more candidates). In this scenario, all LLMs generated correct pairs of task-required tool (manually verified).

Given the importance of ensuring that the LLM is capable of choosing the correct tools for a given task among multiple possible candidates, all self-generated tests were then required to span single tools and pairwise combinations. Note that the native tool use capabilities of the chosen LLMs was used.

The following self-generation methods were proposed and evaluated:

- **use_all**: Given the description of all tools, generate test cases for each tool individually and pairs of tools (all in a single prompt).

- **only_selected**: Given the description of only one or two tools of interest, generate test cases that require using those tools (and nothing else). One prompt per tool / pair was used and the results were aggregated.
- **selected_with_dummies**: Given the description of all tools, generate test cases that require using only one or two tools of interest (and nothing else). One prompt per tool / pair was used and the results were aggregated.

Table 1. Coverage of tests generated by LLMs including single tools and combination of two tools

model	gen_strategy	coverage
Claude 3.5 Sonnet	only_selected	1.00
	selected_with_dummies	1.00
GPT 4o	selected_with_dummies	1.00
	only_selected	0.89
GPT 3.5	selected_with_dummies	0.52
Claude 3 Haiku	only_selected	0.27
GPT 4o	use_all	0.25
Claude 3 Haiku	selected_with_dummies	0.24
Claude 3.5 Sonnet	use_all	0.24
GPT 3.5	only_selected	0.22
Claude 3 Haiku	use_all	0.13
GPT 3.5	use_all	0.11

The results in Table 1 show the coverage of the tests generated, computed as the total number of tests generated per unique tool / pair divided by the total number of unique tool / pairs, $T_C = N_T + \frac{N_T \cdot (N_T - 1)}{2}$, where N_T is the number of tools.

Trying to generate all test cases at once tends not to work, as shown by the poor coverage of the use_all strategy. All LLMs give a compact answer that fails to span the combinations.

Using multiple prompts (T_C in total, one for each unique combination) to generate the test cases enabled better coverage. In particular, including all tool descriptions (selected_with_dummies) proved to be a better strategy both in terms of metrics as well as generating tests of subjective higher quality (manually verified). We conjecture that having access to all tool descriptions helps the LLM be more specific in the test wording.

Note that sometimes the smallest LLMs generate invalid JSON as response. When that happens, the test case is ignored and not added to the list. The most capable models always generated valid JSON.

4.4 LLM Performance Comparison

To verify if LLMs perform better in the test cases generated by them (or their family of models), we set as benchmark and evaluate the tests generated by the best performing models at the time of this paper: Claude 3.5 Sonnet (from Anthropic) and GPT4o (from OpenAI). The generation strategy used in the evaluation was selected_with_dummies, as described in Section 4.3.

We evaluate most pay-as-you-go models available in AWS Bedrock, such as multiple versions of Claude, Mistral, Llama and Command. For the purposes of this evaluation, we left out models considered deprecated by the manufacturers themselves at the time of this study. For example, Llama2 was not included (because it was superseded by Llama3) and Claude 3 Opus was not included (superseded by Claude 3.5 Sonnet). We use the simple prompt “Plan what tool or tools are needed to answer the previous question.” with some extra structure requirements to be able to parse the answers.

Table 2. Metrics of LLMs in the self-evaluation task (364 questions self-generated)

		n_invented_tools	accuracy	score	USD / 1M tokens	
		sum	%	%	Input	Output
model	native_tools					
Claude 3.5 Sonnet	False	0	0.78	0.90	3.00	15.00
GPT 4o	True	1	0.80	0.89	5.00	15.00
GPT 4o mini	True	3	0.80	0.89	0.15	0.60
Claude 3.5 Sonnet	True	0	0.77	0.89	3.00	15.00
Claude 3 Haiku	True	2	0.77	0.89	0.25	1.25
GPT 4o	False	4	0.77	0.88	5.00	15.00
Mistral Large v1	False	1	0.75	0.87	4.00	12.00
GPT 4o mini	False	3	0.73	0.85	0.15	0.60
Command RPlus	False	4	0.73	0.84	3.00	15.00
Claude 3 Haiku	False	3	0.71	0.83	0.25	1.25
GPT 3.5	False	2	0.65	0.79	0.50	1.50
	True	18	0.66	0.77	0.50	1.50
Mistral Mixtral 8x7B	False	156	0.50	0.68	0.45	0.70
Command R	False	117	0.50	0.65	0.50	1.50
Llama3 8b instruct	False	39	0.22	0.38	0.30	0.60
Llama3 70b instruct	False	29	0.29	0.36	2.65	3.50

Table 2 presents metrics to evaluate and decide which model presents the best trade-off in terms of tool choice accuracy, score and pricing. In this evaluation run, a total of 364 questions was self-generated and tested. The column “n_invented_tools sum” shows how many times a particular LLM tried to use a tool that does not exist. . We verify that, contrary to what may be expected, even the LLMs used to generate the test cases were unable to reach 100% accuracy without techiques like chain-of-thought [18] or self-consistency [17]. This result suggests that even current best in class LLMs may be “rushing” to a selection when not explicitly asked to plan actions before answering. However, improving tool selection accuracy is out of the scope of this study.

In terms of the metrics, some notable findings are:

- **Using LLM native tool calling is equal or superior to providing direct tool use instructions**. This is mostly thanks to model providers optimizing their tool use internally for better performance (when compared to our generic instruction designed to work for all LLMs). Claude 3.5 Sonnet may look to be an exception but the native metrics are basically equal, and the manual tool prompting used in this work followed legacy Claude’s model manufacturer (Anthropic) XML guidelines.
- **Larger models perform better**. This is consistent with the widely known fact that increasing the number of parameters improves metrics across all LLM tasks.
- **Closed source models perform better**. Since the closed model training procedure and data is generally undisclosed, and even smaller models like GPT4o mini had acceptable tool calling metrics in an unseen task, it seems likely that private training data contains tool calling samples.

5 CONCLUSIONS

This work introduces a novel method for evaluating Generation Augmented by Tools (GAT) systems, addressing the critical need for task-specific assessment in industrial applications. Our work spans from the theoretical foundations of GAT to practical implementation and evaluation, offering several key contributions to the field. Source code to run GAT with the implemented tools and all tests can be found in https://github.com/douglas125/SelfTestingGAT_LLM.

We start by highlighting the challenges faced by companies in selecting appropriate LLMs for GAT applications, particularly when dealing with proprietary or frequently updated tools. Our research directly addresses this gap, providing a framework for self-evaluation that can adapt to specific industrial contexts.

The provided background establishes the theoretical underpinnings of GAT-LLM architectures and reviews existing evaluation methods. We identified a critical limitation in current benchmarks: their inability to assess performance on custom, industry-specific tools. This finding motivated our development of a self-testing framework.

We design a self-test generation prompt that effectively creates complex, multi-tool test cases. Out of the proposed methods, we find that providing all tool descriptions and explicitly asking LLMs to produce test cases for one or two specific tools provides the best quantitative results (with 100% coverage) and qualitative ones (more complex questions). The implementation of 12 diverse tools, ranging from symbolic math solvers to video manipulation utilities, demonstrated the versatility of our framework.

The results provide empirical evidence of our method's effectiveness. We evaluated LLMs of multiple sizes, both open-source and commercial, on a set of self-generated test cases. Based on those, the best outcome is obtained by using large models with native tool calling capabilities, although some smaller models like GPT4o had competitive results while being more cost effective for this particular set of proposed tools.

In summary, this paper presents a comprehensive approach to GAT evaluation that bridges the gap between theoretical LLM capabilities and practical industrial applications. By providing a flexible, adaptable framework for assessing GAT systems with custom tools, we enable companies to make informed decisions about LLM selection and implementation. This approach allows organizations to balance performance requirements with cost considerations, potentially leading to more efficient and effective AI-augmented workflows across various industries. Our findings on the performance of different LLM sizes and types (open-source vs. commercial) provide valuable insights for practitioners seeking to optimize their GAT systems. Moreover, the self-testing methodology introduced can be readily adapted as new tools are developed or existing ones are updated, ensuring the long-term relevance and effectiveness of GAT applications in dynamic industrial environments.

5.1 Future Work

Future work in this area could expand the scope of evaluation beyond mere function selection accuracy. This could include assessing the correctness of argument passing and the logic behind chaining multiple function calls. Such evaluations would need to be conducted in a context where the specific tools are not known a priori, reflecting the dynamic nature of industrial GAT applications. Additionally, research could explore methods for LLMs to iteratively improve tool descriptions, possibly incorporating human feedback to enhance overall system performance and adaptability. Using human in the loop may be feasible for the evaluation of some GAT solutions.

ACKNOWLEDGMENTS

The author would like to acknowledge the support of SENAI and Instituto SENAI de Inovação - Sistemas Embarcados for the support and willingness to dive into industrial usage of LLMs as a research field.

Thanks also to the colleagues at Verizon Connect Research for the insightful discussions.

REFERENCES

[1] Mistral AI. 2024. *Function Calling - accessed on 2024-07-29.* https://docs.mistral.ai/capabilities/function_calling/

[2] Anthropic. 2024. *Quickstart - accessed on 2024-07-29.* https://docs.anthropic.com/en/docs/quickstart

[3] Anthropic. 2024. *Tool use (function calling) - accessed on 2024-07-29.* https://docs.anthropic.com/en/docs/build-with-claude/tool-use

[4] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2024. A Survey on Evaluation of Large Language Models. *ACM Trans. Intell. Syst. Technol.* 15, 3, Article 39 (mar 2024), 45 pages. https://doi.org/10.1145/3641289

[5] Cohere. 2024. *Single-Step Tool Use - accessed on 2024-07-29.* https://docs.cohere.com/docs/tool-use

[6] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. *NeurIPS* (2021).

[7] Neel Jain, Khalid Saifullah, Yuxin Wen, John Kirchenbauer, Manli Shu, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2024. Bring Your Own Data! Self-Supervised Evaluation for Large Language Models. https://openreview.net/forum?id=zH6zBoktYO

[8] Yu Liu, Duantengchuan Li, Kaili Wang, Zhuoran Xiong, Fobo Shi, Jian Wang, Bing Li, and Bo Hang. 2024. Are LLMs good at structured outputs? A benchmark for evaluating structured output capabilities in LLMs. *Information Processing & Management* 61, 5 (2024), 103809. https://doi.org/10.1016/j.ipm.2024.103809

[9] Jie Lu, Xiangning Tian, Chaobo Zhang, Yang Zhao, Jian Zhang, Wenkai Zhang, Chenxin Feng, Jianing He, Jiaxi Wang, and Fengtai He. 2024. Evaluation of large language models (LLMs) on the mastery of knowledge and skills in the heating, ventilation and air conditioning (HVAC) industry. *Energy and Built Environment* (2024). https://doi.org/10.1016/j.enbenv.2024.03.010

[10] Nestor Maslej, Loredana Fattorini, Raymond Perrault, Vanessa Parli, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, and Jack Clark. 2024. Artificial Intelligence Index Report 2024. arXiv:2405.19522

[11] OpenAI. 2024. *Function calling - accessed on 2024-07-29.* https://platform.openai.com/docs/guides/function-calling

[12] Qiwei Peng, Yekun Chai, and Xuhong Li. 2024. HumanEval-XL: A Multilingual Code Generation Benchmark for Cross-lingual Natural Language Generalization. arXiv:2402.16694 [cs.CL] https://arxiv.org/abs/2402.16694

[13] Zafaryab Rasool, Stefanus Kurniawan, Sherwin Balugo, Scott Barnett, Rajesh Vasa, Courtney Chesser, Benjamin M. Hampstead, Sylvie Belleville, Kon Mouzakis, and Alex Bahar-Fuchs. 2024. Evaluating LLMs on document-based QA: Exact answer selection and numerical extraction using CogTale dataset. *Natural Language Processing Journal* 8 (2024), 100083. https://doi.org/10.1016/j.nlp.2024.100083

[14] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. arXiv:2311.12022 [cs.AI] https://arxiv.org/abs/2311.12022

[15] Konstantinos I. Roumeliotis, Nikolaos D. Tselikas, and Dimitrios K. Nasiopoulos. 2024. LLMs in e-commerce: A comparative analysis of GPT and LLaMA models in product review evaluation. *Natural Language Processing Journal* 6 (2024), 100056. https://doi.org/10.1016/j.nlp.2024.100056

[16] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL] https://arxiv.org/abs/2307.09288

[17] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. arXiv:2203.11171 [cs.CL] https://arxiv.org/abs/2203.11171

[18] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL] https://arxiv.org/abs/2201.11903

[19] Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley Function Calling Leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html.

Received 20 July 2024; revised -; accepted -