

Building and Evaluating a Math Agent using LangChain

Andre Ngo

Institute for Informatics

Ludwig-Maximilians-Universität München

Munich, Germany

Laura Bengs, B. Sc.

Institute for Informatics

Ludwig-Maximilians-Universität München

Munich, Germany

Abstract—Agents are an emergent field in AI, which is concerned with equipping large language models (LLMs) with further capabilities such as tool calling, thereby eliminating the need for users to orchestrate individual steps and extending the capabilities of LLMs beyond conventional chatbots. In this work, we build a Math Agent using LangGraph, which is able to solve mathematical problems using two tools: a document extractor and a calculator. Evaluating four different models, namely Qwen 2.5-7B, Qwen 2.5-14B, Claude 3.5 Sonnet and Claude 3.7 Sonnet on the GSM8K dataset, we derive the following key findings: 1) Enabling tool calling improves the correctness of LLMs. 2) Lower correctness of LLMs due to a smaller model size can be equalized with adequate tool calling. 3) On the flipside, Tool calling increases latency as well as token usage. Furthermore, we are able to show that our implementation can keep up with current benchmarks.

I. INTRODUCTION

Large language models (LLMs) have been the subject of intense research and development over the past few years. However, as their capabilities grow, common issues such as hallucinations, inconsistent results, and substantial resource usage persist. This is especially noticeable when using it for multi-step, complex mathematical calculations. Since LLMs are fundamentally relying on an autoregressive transformer-based architecture, predicting the next-most likely token, their actual understanding is limited.

Providing LLMs with classical symbolic math engines could help mitigate these problems. Instead of converting math expressions to higher-dimensional vectors, these traditional approaches could deliver more reliable and deterministic results. The recent emergence of “agentic” artificial intelligence (AI) is useful here, enabling LLMs to make use of such tools to compute results autonomously without requiring human intervention.

We hypothesize that augmenting an LLM’s capabilities by providing such tools will increase the accuracy of outputs. Since transformer-based computations have proven to be resource intensive, offloading some of the compute to highly optimized Python libraries could also potentially come at a performance benefit.

II. FOUNDATIONS

A. Agentic AI

At the forefront of recent developments in AI is the emergence of so-called **Agents**. The terminology refers to the agency of AI systems (usually a LLM) to take autonomous decisions of making use of various tools, services, and applications available to accomplish certain tasks on behalf of the user. Thereby, the need for users to orchestrate individual steps by hand is eliminated, extending the capabilities of LLMs beyond conventional chatbots.

Crucially, the LLM retains the role of **tool calling**, meaning it has the sole responsibility to plan and choose the usage of the given tools, passing the necessary input and determining the next steps based on the tool’s output. This is a key distinction from simple, predetermined workflows.

B. Framework ecosystem

The math agent’s architecture is built upon **LangChain**, an open-source framework for developing LLM applications, integrating with several tools of the ecosystem such as LangGraph and LangSmith. It provides modularity in regard to the usage of different tools, models and inference providers. With **LangGraph**, a graph-based architecture is used to model and manage the relationships between various components of an AI agent, allowing more control and refinement. Furthermore, **LangSmith** serves as a platform for debugging, testing, evaluating, and monitoring LLM applications by providing detailed traces of test runs.

C. Inference providers

Ollama is an open source tool for running LLMs locally on machine. Model weights, configuration, and data are bundled into a single package, and furthermore, customizable optimizations for inference, such as quantization and choosing the context length, are available.

When using a closed-source model, the API infrastructure of the respective company behind it was used.

III. SYSTEM DESIGN

The core idea of this project is to build an agent that is able to solve mathematical equations. Given a document (e.g.

as PDF, Word, Image) the agent should extract mathematical tasks, use a symbolic calculator and then return the solution to the user. Thereby, the LLM should make use of two tools: a document extractor and a Python math calculator.

A. Document extractor

Docling [1] is an open source Python library by IBM released in 2024. It is a lightweight, efficient document parser that supports various formats such as PDF, images, docx and HTML. Additionally, it has extensive support of various OCR engines, such as EasyOCR (by default), Tesseract or RapidOCR, to name a few, in order to support formula recognition and text in images. It also supports GPU acceleration via Nvidia CUDA. We used Docling to implement a document extractor tool, that is able to receive a file as input and return the content of the file in markdown.

B. Math calculator

For the implementation of the calculator tool we used **NumExpr** [2], a fast numerical array expression evaluator for NumPy. Compared to using pure Python, it accelerates calculations while also using less memory. It also has the advantage of being able to evaluate an expression within a single function, saving the LLM from having to orchestrate further tool calls (i.e. a different tool for every single mathematical operation) which could hamper performance.

A potential caveat of NumExpr is that it might introduce overhead for very simple arithmetic calculations (e.g. $1+1$). However, considering the waterfall traces (such as in Fig. 3 and 4), this has proven not to be the bottleneck of our project and as such had negligible to no negative impact on the performance of the math agent for these calculations.

C. Large Language Models

Different **LLM**'s came into consideration when implementing the math agent (also see Section VI for more information on the tested models). We wanted to compare current publicly available models that can be run locally on a machine, as well as closed source models accessible via an API. For all models, a requirement was to explicitly support tool calling. Encountering resource limitations, we restricted the choice of locally-run models to those with less than ~ 16 billion parameters. Furthermore, the choice of API models was economically constrained, as only those with a free tier came into consideration. In the end, we chose the following models for multiple reasons:

- Qwen 2.5-7B (open weights, Jan. 2025)
- Qwen 2.5-14B (open weights, Jan. 2025)
- Claude 3.5 Sonnet (proprietary, Oct. 2024)
- Claude 3.7 Sonnet (proprietary, Feb. 2025)

For the Claude models, Anthropic granted us a \$50 credit for their API as part of their student program. Although Google's API offers a free tier for their Gemini models with different rate limits, we were not able to use their models. The model with the most lenient rate limits, Gemini 2.0 Flash-Lite

(proprietary, Feb. 2025), was successful on individual runs, but an evaluation on the dataset was not possible due to quickly exceeded rate limits.

For all models, the system prompt provides the LLM with information on how to approach a given task in general, giving important instructions and constraints to help generate the desired output. We had the following system prompt, which was changed but remained the same for each prompt made: "You are a helpful agent tasked to solve mathematical problems. If a file is provided: use the 'pdfFormula' tool first to extract the math problems. Then: use the calculator tool to compute the answer to the problem. If there are multiple problems: return the answer for each problem. You are not allowed to calculate things yourself but have to use the calculator tool. Return the solution from the calculator tool, *DONT* change the answer. *ONLY* return the solution(s) (should be only a number; no metrics for each task). *DONT* repeat the instruction. *DONT* answer in complete sentences."

D. Architecture overview

Modeling the math agent in LangGraph, the architecture is as follows: the LLM is connected to the PDF extractor tool and the calculator tool. When starting the math agent the LLM is called and will take over the routing. At any given moment the LLM can decide whether it wants to call a tool, and therefore continue, or if the task is completed and the process should terminate. If a tool is called, the LLM has to wait until the calculations took place and results are available. Fig. 1 illustrates this set up.

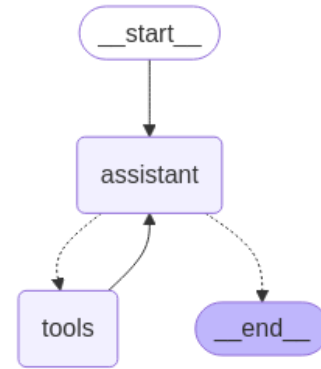


Fig. 1. Directed Graph modeling the agent's architecture. Full edges indicate fixed transitions; dotted edges indicate conditional transitions, over which the LLM (here: the assistant) has control.

IV. EXAMPLE RUNS

A. Single mathematical task

The following example run illustrates the usage of the math agent. In a PDF file (named question.pdf) the task "An ice cream factory makes 100 quarts of chocolate ice cream in 2 hours. It can make 50 quarts of vanilla ice cream in 4 hours. How many quarts in total would be made in 48 hours?" was

saved. The math agent was invoked with the prompt "What is the solution for content/question.pdf?", only passing over the filepath embedded in a natural question.

```

===== Human Message =====
what is the solution for /content/question.pdf
===== Ai Message =====
Tool Calls:
pdfFormula (0ae7afdf-a196-495f-a217-e3ed7e099001)
Call ID: 0ae7afdf-a196-495f-a217-e3ed7e099001
Args:
  filepath: /content/question.pdf
===== Tool Message =====
Name: pdfFormula
An ice cream factory makes 100 quarts of chocolate ice cream in 2 hours. It can make
50 quarts of vanilla ice cr 4 hours. How many quarts in total would be made in 48
hours?
===== Ai Message =====
Tool Calls:
calculator (175faa9f-24b7-4cb3-a2c0-7115f5ba954c)
Call ID: 175faa9f-24b7-4cb3-a2c0-7115f5ba954c
Args:
  expression: (100/2 + 50/4) * 48
===== Tool Message =====
Name: calculator
3000.0
===== Ai Message =====
Task 1: 3000

```

Fig. 2. Output of math agent after asking what the solution for question.pdf is

In the first step, the LLM decided to make a tool call of the document extractor. The document extractor conceived the PDF file as input, extracted the content and returned it to the LLM. The LLM then decided to make another tool call. After converting the task into a mathematical expression, the calculator tool was called to obtain the solution. Afterward, the LLM decided enough steps were taken to solve the task appropriately, and returned the solution to the user.

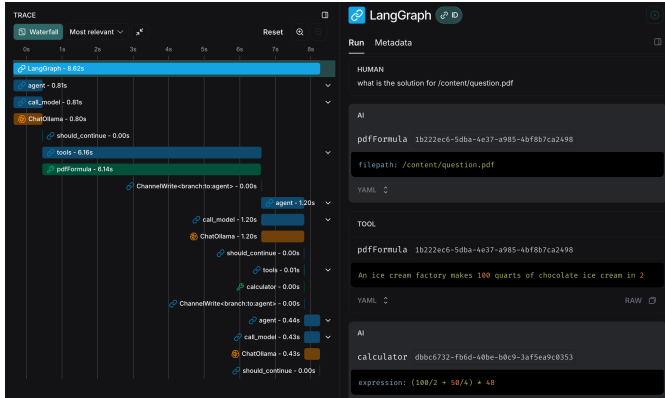


Fig. 3. Trace of the example run using LangSmith

A trace of this run can be evaluated using LangSmith, also shown in Figure 3. Here, the sequence of the run is visualized as a waterfall diagram, where information, such as the time it took to execute a certain step, is visible. It reveals that document extraction (here named "pdfFormula") with 6.14s took the most amount of time, whereas the math calculator was called and finished computation within a fraction of a second.

B. Multiple mathematical tasks

In contrast to the previous example, this time three different mathematical tasks were saved in one PDF file. Through this, the math agent was forced to first extract all three problems, convert them into mathematical expressions suitable for the calculator tool, make three calculator tool calls and then return the solutions to the user. As can be seen in Fig. 4 the math agent was able to accomplish the task by making multiple tool calls, deciding correctly when to stop and returning the solutions to the user.

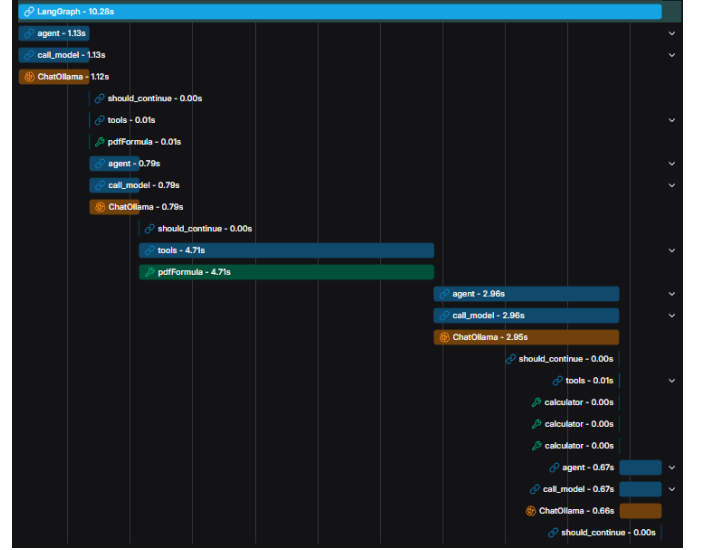


Fig. 4. Solving a PDF with multiple mathematical tasks using multiple tool calls

V. EVALUATION

A. Dataset

For the evaluation, the **GSM8K** dataset was used, which serves as a common benchmark for current models. As per the paper's abstract, it is a dataset of "8500 linguistically diverse grade school math word problems created by human writers. Each problem takes between two and eight steps to solve, only requiring basic arithmetic operations (e.g. +, -, ×, ÷). It is designed so that a bright middle school student is able to solve every problem." [3]

The data set was slightly adapted to fit our needs, as cleanup on the answer columns was performed to only include the numeric value instead of being embedded in the answer sentences. Furthermore, we ran our evaluation on a subset of 100 data points of the original dataset, to be able to test multiple different setups and configurations.

B. Test environment

As test environment, a Jupyter Notebook was run on a GPU runtime hosted on Google Colab. As such, 12.7 GB of RAM and 15.0 GB GPU memory on a NVIDIA Tesla T4 were available. It is to be noted that runtime and compute

allocation on the free tier is subject to fluctuations [4], though once granted, this should not have affected the performance on individual runs.

C. Parameters

When setting up the models, a baseline of parameters was used considering context length, maximum output token length and temperature in order to ensure comparability.

- Context length: 4096 tokens
- Temperature: 0 (lowest possible)
- Output length: unrestricted (maximum possible)

The **context length**, also known as context window, refers to the amount of tokens a model is capable of processing in a single input sequence, impacting how much information a model can consider at once (representing a "working memory"). This includes input and output tokens. For the Qwen 2.5 models, the maximum context length is 128k [5] and for Claude 3.5 [6] and 3.7 [7] it is 200k. However, since computing resources for the local models were limited, the context length for all models was restricted to 4096 tokens.

Adjusting the **temperature** setting affects the amount of "creativity" and randomness of a model's response. This works by refining the probability distribution on which the next token is predicted. When using a temperature close to 0, tokens with the highest probability are much preferred over lower ones, resulting in more deterministic outputs. This setting is especially useful in contexts for mathematical tasks and tool calling, where consistency and dependable results are essential.

Output length refers to the maximum amount of tokens a model is allowed to generate. In order to compare a model's efficiency regarding the generation of tokens to solve a task (i.e. how many tokens are needed to come up with a solution), setting apart different architectures and design philosophies, the parameter was set to unrestricted. This is especially interesting due to the expectation that tool calling would vastly increase the amount of tokens generated, caused in great part by the additional steps involved.

D. Evaluation with PDF input

As the GSM8K dataset contains questions in string format, we wrote a short code snippet to generate a PDF file with the question as content for each datapoint in the dataset. Through that we were able to test the performance of the MathAgent with PDF inputs.

Correctness: As can be seen in Table I, the highest correctness was achieved with Claude 3.5 Sonnet and Claude 3.7 Sonnet, which each scored a value of 0.85. Using Qwen-2.5, a significantly smaller correctness was achieved with a value of 0.57 for 7B parameters and 0.12 for 14B parameters. We hypothesize that the distinctive higher correctness of answers when using Claude 3.5 Sonnet and Claude 3.7 Sonnet is related to the size of the models (speculated to be

at $\sim 175B$ parameters [8]). In our evaluation, bigger models usually performed significantly better compared to models with a smaller number of parameters. A later evaluation on string inputs in Section V-E supports this hypothesis.

Model	Correctness	P50 Latency	P99 Latency
Claude 3.5 Sonnet	0.85	13.66s	22.77s
Claude 3.7 Sonnet	0.85	12.58s	23.48s
Qwen 2.5-7B	0.57	7.70s	16.77s
Qwen 2.5-14B	0.12	10.80s	20.68s

TABLE I. Evaluation on 100 datapoints of the GSM8K dataset converted into PDF files

Latency: While Qwen-2.5 was not able to compete with Claude 3.5 Sonnet and Claude 3.7 Sonnet regarding correctness, the model has a lower latency and results were made available to the user more quickly. 50 % of our request made to Qwen 2.5-7B were processed in less than 7.70 seconds (Qwen 2.5-14B: 10.80 seconds) and 99 % of our requests were processed faster than 16.77 seconds (Qwen 2.5-14B: 20.68 seconds). Claude 3.5 Sonnet and Claude 3.7 Sonnet record slightly longer times with P50 latencies of 13.66 seconds and 12.58 seconds and P99 latencies of 22.77 seconds and 23.48 seconds.

In conclusion, using one of these tested models a compromise between correctness and latency has to be made: Qwen 2.5-7B and Qwen 2.5-14B make results available faster than Claude 3.5 Sonnet and Claude 3.7 Sonnet thereby compromising on correctness, while the Claude models return solutions with a higher accuracy but taking significantly longer to answer.

Additionally to evaluating the math agent (e.g. the different models with tool calling enabled) we tried to invoke the models with the same prompt but without tool calling enabled. However, this failed as Ollama does not support PDF input when used through LangChain [9]. In order to be able to make a comparison between models with tool calling enabled and models without tool calling enabled, we therefore decided to run another evaluation on string inputs instead of PDF files as input.

E. Evaluation with string input

In this evaluation we used the string inputs from the GSM8K dataset. In the following "with tool calling" corresponds to the models being able to use the calculator tool whereas "without tool calling" considers the model on its own without further capabilities. Table II gives an overview of the most interesting runs.

Correctness: As previously hypothesized, larger models performed significantly better than smaller models with respect to the correctness of solutions. While our smallest tested model (namely Qwen 2.5-7B) achieved a correctness of 0.24 (without tool calling) and 0.59 (with tool calling), Claude 3.5 Sonnet achieved a correctness of 0.60 (without tool calling)

and 0.95 (with tool calling). This is a significantly higher level of correctness and nicely illustrates the relationship between model size and correctness. Thus, we find it interesting to note that Qwen 2.5-7B with tool calling enabled performed better (correctness: 0.59) than Qwen 2.5-14B without tool calling (correctness: 0.32). This indicates that a lower accuracy of the agent due to a smaller model size can be equalized with adequate tools.

Model	Tool Calling	Correctness	P50	P99
Claude 3.5 Sonnet	True	0.95	4.08s	11.18s
Claude 3.5 Sonnet	False	0.60	0.52s	1.29s
Claude 3.7 Sonnet	True	0.92	5.45s	14.41s
Claude 3.7 Sonnet	False	0.68	0.73s	1.53s
Qwen 2.5-14B	True	0.46	0.38s	5.34s
Qwen 2.5-14B	False	0.32	0.31s	0.55s
Qwen-7B	True	0.59	1.31s	6.60s
Qwen-7B	False	0.24	0.16s	0.43s

TABLE II. Evaluation on 100 datapoints of the GSM8K dataset

Tool calling: As illustrated in Fig. 5, the math agent performed significantly better if tool calling was enabled (blue beam). Using Qwen 2.5-14B the correctness improved by 0.14 points, in Claude 3.7 by 0.24 points and in Claude 3.5 as well as in Qwen 2.5-7B an improvement each by 0.35 points can be recognized if tool calling was enabled. On the other hand, we need to mention that if tool calling is enabled, requestes were processed distinctivtly slower. To give an example: Qwen 2.5-B was able to process requests in less than 0.16 seconds in 50 % of all cases and in less than 0.43 seconds in 99 % of all cases. When tool calling was enabled Qwen 2.5-7B took significantly longer and needed at most 1.31 seconds in 50 % of all cases and 6.60 seconds in 99 % of all cases. This means looking at P50 latency an increase of 1.15 seconds was recorded and for P99 latency the increase amounts to 6.17 seconds.

Latency: In Section V-D, the larger models recorded higher latencies compared to the smaller models. In this evaluation of string inputs, results aligned with the observation: While regarding only the models without tool calling enabled, the P50 and P99 latencies were not too far apart (P50: in the range of 0.16-0.73 seconds; P99: in the range of 0.43-1.53 seconds), although even then the models we ran locally (namely Qwen 2.5-7B and Qwen 2.5-14B) registered lower latencies. When tool calling was enabled, this gap widened as Qwen 2.5-7B and Qwen 2.5-14B were able to return solutions faster than Claude 3.5 Sonnet and Claude 3.7 Sonnet.

Regarding P50 latency, 50% of the requests made to Qwen 2.5-14B were processed faster than 0.38 seconds (Qwen 2.5-7B: 1.31 seconds), requests made to Claude 3.5 Sonnet took 4.08 seconds, and for Claude 3.7 Sonnet, even 5.45 seconds. A similar picture emerges when looking at P99 latencies: While 99% of the requests made to Qwen 2.5-14B

were processed faster than 5.34 seconds (Qwen 2.5-7B: 6.60 seconds), the requests made to Claude 3.5 Sonnet took 11.18 seconds (Claude 3.7 Sonnet: 14.41 seconds). Looking at Fig. 6, we can furthermore see that enabling tool calling has a large impact on the latency, as all models took significantly longer for calculations. For Claude 3.7 Sonnet the P99 latency increased by a factor of 9.41 when tool calling was enabled, meaning that the model took almost 10x as long to provide answers to user input.

Token count: Another important indicator when comparing the performance of LLMs is the amount of tokens needed for generating a result. Table III showcases the considerable differences in the amount of tokens across the two model families accumulated when executing the runs on the dataset, notably when regarding runs with tool calling enabled. As expected, tool calling led to a massive increase in tokens generated ($\sim 24x$ more for Claude 3.5 and 3.7 Sonnet, $\sim 6x$ for Qwen 2.5-14B and $\sim 10x$ for Qwen 2.5-7B), as visualized in Fig. 7. This is likely due to multiple things such as understanding the purposes of the tools provided, additional steps involved in calling the tools, considering its output and planning the next actions based on it.

Model	Tool Calling	Total Tokens
Claude 3.5 Sonnet	True	273882
Claude 3.5 Sonnet	False	11463
Claude 3.7 Sonnet	True	275817
Claude 3.7 Sonnet	False	11473
Qwen 2.5-14B	True	67871
Qwen 2.5-14B	False	11357
Qwen 2.5-7B	True	111769
Qwen 2.5-7B	False	11344

TABLE III. Cumulative token count per model

Strikingly, Anthropic’s Claude models are much more lavish in their usage of tokens compared to Alibaba’s Qwen models ($\sim 4x$ and $\sim 2.5x$ increase compared to Qwen 2.5-14B and Qwen 2.5-7B with tool calling enabled, respectively), owing in part to the vastly bigger parameter count. Interestingly though, token count across all models without tool calling remained at about $\sim 11,300$ irrespective of model size, mitigating the argument that more parameters equal more tokens generated. Thus, we hypothesize that Claude’s architectural implementation of tool calling is more considerate (i.e. “thinking more”) in using the tools provided, involving more steps in understanding the solution using the tools, resulting in the higher correctness score we observed.

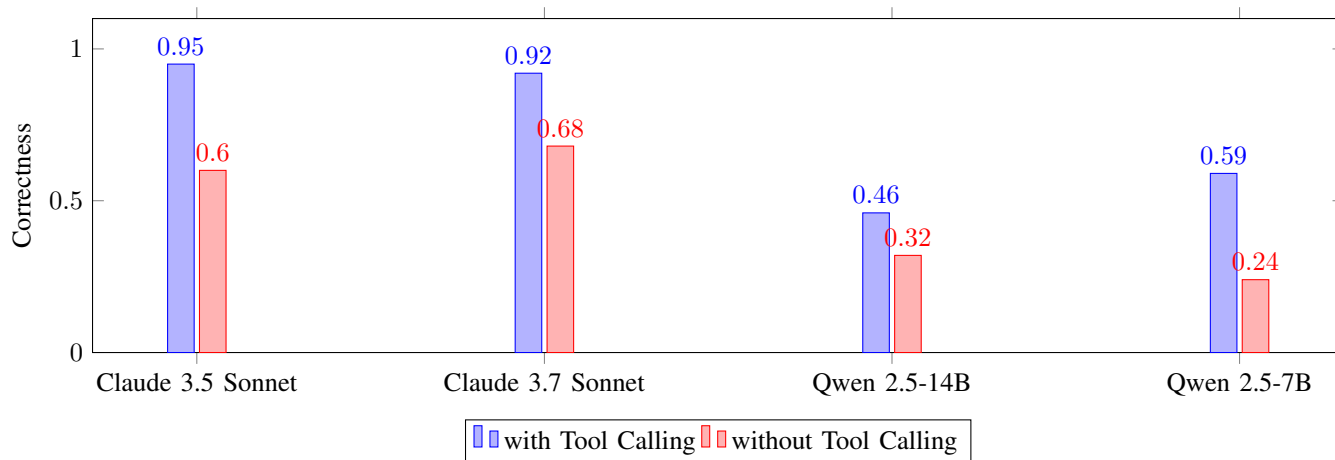


Fig. 5. Correctness per Model (with/without Tool Calling)

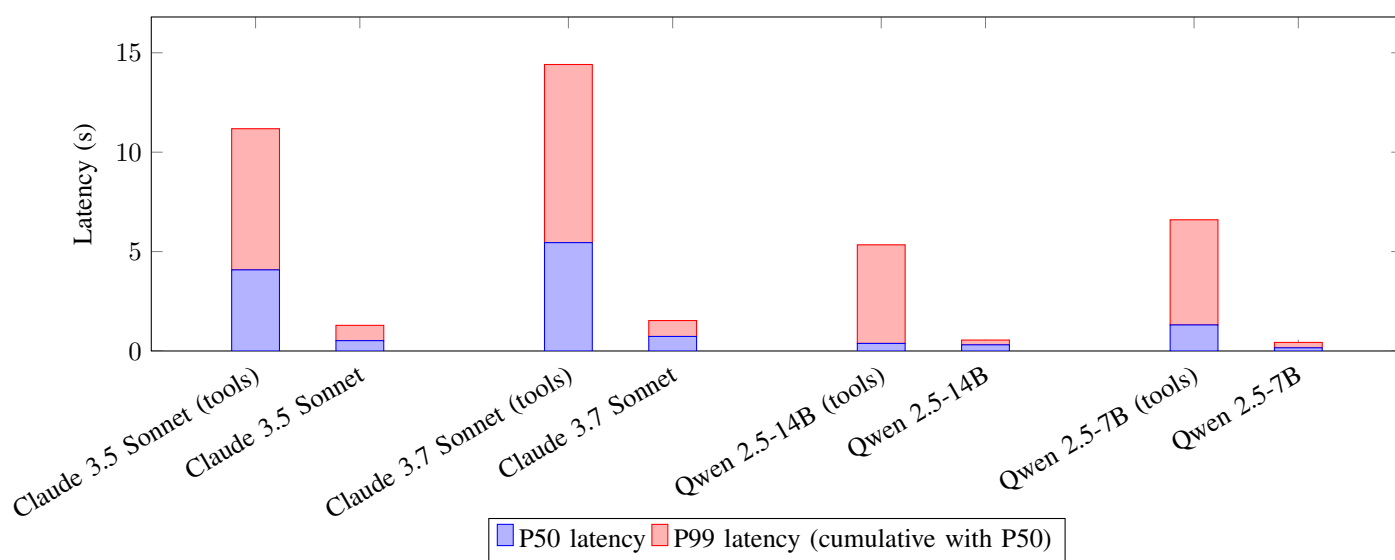


Fig. 6. P50 and P99 Latencies by Model (with/without Tool Calling)

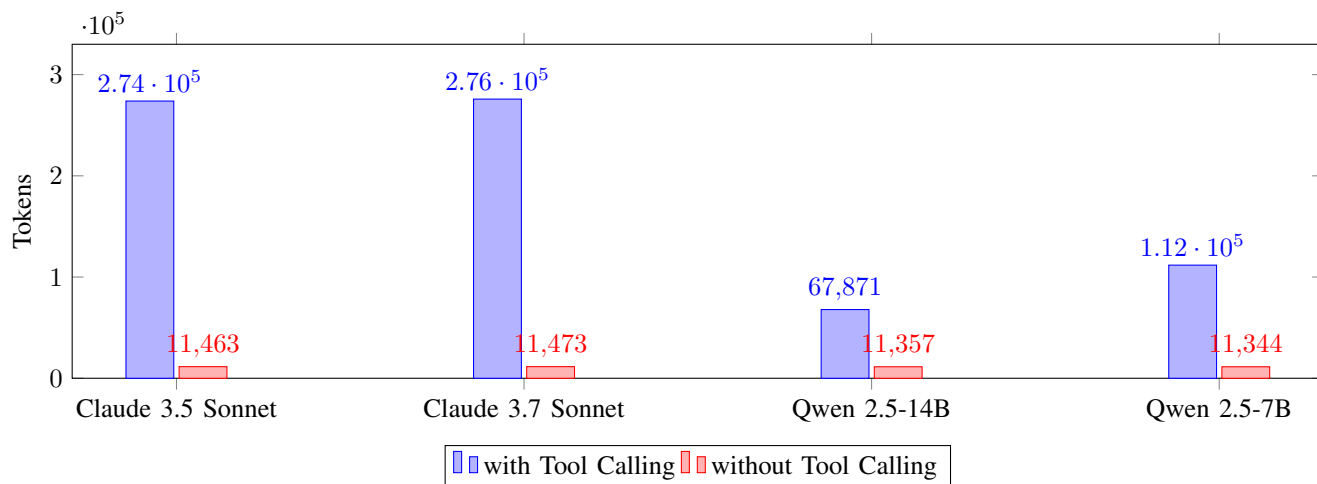


Fig. 7. Cumulative token count by model (with/without Tool Calling)

This is also likely the explanation for the massively increased latency for the Claude models compared to Qwen 2.5 when using tool calling. Therefore, it is probable that latency introduced by the nature of API calls compared to running a model locally can be viewed as negligible, especially when considering that Anthropic likely has far more computing power available than our Google Colab environment.

F. Comparison to GSM8K benchmarks

As mentioned before in Section V-A, the GSM8K dataset is a widely available benchmark [10] for current models, enabling a comparison to our findings.

Claude 3.5 Sonnet: One benchmark conducted with Claude 3.5 Sonnet (with hierarchical prompting) [11] currently ranks first as it achieves the highest accuracy of all models with 97.72. While in our evaluation the model without tool calling (correctness: 0.60) was not able to compete with this value, the model with tool calling enabled (correctness: 0.95) does get near the set benchmark of the model. Comparing our results to other models in the benchmark database, our evaluation of Claude 3.5 Sonnet without tool calling would rank 117th and Claude 3.5 Sonnet with tool calling would be ranked 8th in the list next to Claude 3 Opus, which also achieved an accuracy of 0.95.

Claude 3.7 Sonnet: Claude 3.7 Sonnet is currently not included in the benchmark. Regarding the correctness values we achieved when we tested the model, it would rank 107th without tool calling and with tool calling 16th together with AlphaLLM [12].

Qwen 2.5: While the model with 7B parameters ranks 148th without tool calling and 119th with tool calling enabled, the model with 14B parameters ranks 147th when regarding results without tool calling and 137th when tool calling is enabled.

It should be noted that an equal comparison between these benchmarks is not given, since testing methodologies can differ a lot. For instance, some of the tests were conducted using few-shot training, extra training data, or chain-of-thought-reasoning, which would improve the accuracy scores, thus not making it a fair comparison to our tests.

VI. CHALLENGES

A. LLM choice

The choice of a suitable model was not as trivial as initially assumed. Although we made sure that the selected models support tool calling, we encountered multiple problems during testing. Some models (e.g., Llama 3.2 3B, Watt-Tool 8B) were unable to determine appropriate next steps and/or call the given tools correctly, which either

resulted in an endless loop or did not produce a result at all. Other models, like Hermes-2-Theta and Llama-3-8B, were not able to accurately respond to our prompts and started to hallucinate instead. Resorting to bigger models (e.g., with more than ~ 16 billion parameters) might resolve some of the issues, but running them reliably was not feasible for us.

Regarding our capabilities and resources, we were not able to run multiple models, such as Watt-Tool-70B or Llama-3.3-70B, as these models have too many parameters and are computing-intensive. This is why we resorted to models with less than ~ 16 billion parameters when running locally, thereby restricting our choice to much smaller models and not being able to test the whole range of models offered. Models accessed through an API offered a possibility for us to test bigger models without overloading our computing capacity. This worked well for the Claude models, as Anthropic granted us a \$50 credit for their API as part of their student program. However, when testing models from Google (e.g., Gemini-1.5-Pro, Gemini-2.0-Flash-Lite, Gemini-2.0-Flash-Experimental), we quickly encountered rate limits that restricted our usage time and made an evaluation on 100 datapoints impossible.

B. Google Colab

Another problem corresponds to our selected programming environment, Google Colab. The limited compute and arbitrary runtime allocation sometimes made it impossible to test new implementations and get a reliable evaluation. Additionally, Docling’s GPU usage on complex PDF files combined with running the LLM locally crashed the session occasionally. By using multiple different user accounts, we managed to build a workaround and were able to test the different LLMs with our math agent. For future work it would be beneficial to switch to another programming environment, such as the resources offered by LRZ Cloud Compute.

C. LangSmith

Similar to Google Colab, we used LangSmith on a free tier. This means we were only able to make 5000 free traces per month and were then asked to set up a payment method in order to be able to continue. We did consider using Langfuse [13] instead, as their offer includes 50k events/month, but then ended up finding a workaround through a new user account that made it possible to continue our evaluation with LangSmith.

VII. DISCUSSION

A. Limitations

Regarding the GSM8K dataset, while it is still very much relevant in current benchmarks, it is only available in English. As this is the language in which most training data is available, mathematical solving abilities for tasks in other languages are ignored.

Furthermore, due to resource limitations, we ran the evaluation on only 100 datapoints. While this is likely to be representative enough on its own, this could hamper comparisons to other benchmark tests that were run on the entire dataset. Additionally, as mentioned in V-F, we were not exactly able to reproduce publicly available benchmarks, owing to differences in setup.

Lastly, in our test runs, the maximum output length was unrestricted, which likely enabled the massive difference in token usage. Setting fixed limits on tokens generated might give indications on how well a model performs per token count (i.e., accuracy points per token).

B. Future work

For future research, equipping smaller language models (i.e., with less than 3B parameters) with math tools could be of interest since our findings showed that Qwen 2.5-7B with tool calling enabled proved to be competitive, outperforming its 14B equivalent (even with tool calling) in accuracy. In addition, prompt engineering (e.g. Chain-of-Thought, Few-shot prompting) as employed in other tests, could further enhance correctness. Also, the performance deficit of tool calling could potentially be mitigated if math engines were supported natively by the architecture of LLM's, instead of the resource-intensive abstraction using tokens to call a tool.

VIII. CONCLUSION

In conclusion, the following statements can be made:

- 1) Larger models perform significantly better regarding correctness but, on the otherhand, take longer for calculations.
- 2) Smaller models are able to return the solution much faster, but thereby compromise on the correctness.
- 3) Enabling tool calling enhanced the performance of our math agent as the correctness improved over all considered models.
- 4) The lower correctness of smaller models can be equalized with adequate tools, thereby achieving even higher correctness than larger models without tool calling.
- 5) A downside of enabling tool calling is the increase in latency, meaning users get their answers later compared to models that do not use tools.

Our findings provide evidence that augmenting LLMs with traditional, symbolic math engines does improve the accuracy of results significantly. This could potentially be a contributing new technique for improving AI reliability. On the downside, it introduces a significant latency and computing resource penalty, necessitating the usage of much more tokens than without tool calling, hence we were not able to verify our hypothesis of yielding a performance improvement. Nevertheless, this could possibly be mitigated if symbolic math engines were to be deeply embedded in the architecture of LLMs, making usage more token-efficient.

Lastly, analogous to humans employing calculators to yield reliable and dependable results and off-load cognitive capabilities, freeing up resources for other demanding tasks, the same could be done for LLMs using traditional, time-tested, symbolic math libraries – perhaps, you must not always reinvent the wheel.

ACKNOWLEDGMENT

Special thanks to Dr. Andre Luckow for teaching the Advanced Analytics and Machine Learning course (AAML) at LMU Munich. His lectures provided deep insight into state-of-the-art technologies and recent advancements, whereas the corresponding coursework and group project gave the students valuable hands-on experience.

REFERENCES

- [1] N. Livathinos, C. Auer, M. Lysak, A. Nassar, M. Dolfi, P. Vagenas, C. B. Ramis, M. Omenetti, K. Dinkla, Y. Kim, S. Gupta, R. T. de Lima, V. Weber, L. Morin, I. Meijer, V. Kuropiatnyk, and P. W. J. Staar, "Docling: An efficient open-source toolkit for ai-driven document conversion," 2025.
- [2] R. McLeod, F. Alted, A. Valentino, G. de Menten, M. Wiebe, cgohlke, A. Bedini, mamrehn, anatoly techtonik, S. Erb, A. Shadchin, A. Bunin, T. Kooij, O. Pavlyk, M. B. Jelloul, L. Garrison, D. M. Hurtado, C. Carey, M. Sarahan, D. Cox, Z. Plesivčák, J. Borgdorff, C. Courbet, M. Dickinson, B. Leitao, A. de Laat, A. Pitrou, A. Portnoy, A. L. Ortega, and A. Böhn, "pydata/numexpr: Numexpr v2.6.9," Dec. 2018.
- [3] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training verifiers to solve math word problems," *CoRR*, vol. abs/2110.14168, 2021.
- [4] Google, "Colaboratory - frequently asked questions." Accessed: 2025-04-24.
- [5] Qwen, :, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu, "Qwen2.5 technical report," 2025.
- [6] Anthropic, "Claude 3.5 sonnet model card addendum," 2024. Accessed: 2025-04-27.
- [7] Anthropic, "Claude 3.7 sonnet system card," 2025. Accessed: 2025-04-27.
- [8] A. B. Abacha, W. wai Yim, Y. Fu, Z. Sun, M. Yetisgen, F. Xia, and T. Lin, "Medec: A benchmark for medical error detection and correction in clinical notes," 2025.
- [9] I. LangChain, "Chatollama," 2025. Accessed: 2025-04-28.
- [10] M. AI, "Arithmetic reasoning on gsm8k." Accessed: 2025-04-27.
- [11] D. Budagam, A. Kumar, M. Khoshnoodi, S. KJ, V. Jain, and A. Chadha, "Hierarchical prompting taxonomy: A universal evaluation framework for large language models aligned with human cognitive principles," 2024.
- [12] Y. Tian, B. Peng, L. Song, L. Jin, D. Yu, H. Mi, and D. Yu, "Toward self-improvement of llms via imagination, searching, and criticizing," 2024.
- [13] L. G. . F. T. Inc., "Langfuse," 2022-2025. Accessed: 2025-04-28.