LLM-Based Code Generation for Querying Temporal Tabular Financial Data

Mohamed Lashuel
Department of Mathematical Sciences
Rensselaer Polytechnic Institute
Troy, NY, USA
lashum@rpi.edu

Gulrukh Kurdistan
Department of Management
Rensselaer Polytechnic Institute
Troy, NY, USA
kurdig@rpi.edu

Aaron Green
Department of Mathematical Sciences
Rensselaer Polytechnic Institute
Troy, NY, USA
greena12@rpi.edu

John S. Erickson
The Future of Computing Institute at Rensselaer
Rensselaer Polytechnic Institute
Troy, NY, USA
erickj4@rpi.edu

Oshani Seneviratne
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY, USA
senevo@rpi.edu

Kristin P. Bennett
Department of Mathematical Sciences
Rensselaer Polytechnic Institute
Troy, NY, USA
bennek@rpi.edu

2024 IEEE Symposium on Computational Intelligence for Financial Engineering and Economics (CIFEr) | 979-8-3503-5483-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/CIFER62890.2024.10772804

***Abstract*—We examine the question of "how well large language models (LLMs) can answer questions using temporal tabular financial data by generating code?". Leveraging advanced language models, specifically GPT-4 and Llama 3, we aim to scrutinize and compare their abilities to generate coherent and effective code for Python, R, and SQL based on natural language prompts. We design an experiment to assess the performance of LLMs on natural language prompts on a large temporal financial dataset. We created a set of queries with hand-crafted R code answers. To investigate the strengths and weaknesses of LLMs, each query was created with different factors that characterize the financial meaning of the queries and their complexity. We demonstrate how to create specific zero-shot prompts to generate code to answer natural language queries about temporal financial tabular data. We develop specific system prompts for each language to ensure they correctly answer time-oriented questions. We execute this experiment on two LLMs (GPT-4 and Llama 3), assess if the outputs produced are executable and correct, and assess the efficiency of the produced code for Python, SQL, and R. We find that while LLMs have promising performance, their performance varies greatly across the languages, models, and experimental factors. GPT-4 performs best on Python (95.2% correctness) but has significantly weaker performance on SQL (87.6% correctness) and R (79.0% correctness). Llama 3 is less successful at generating code overall, but it achieves its best results in R (71.4% correctness). A multi-factor statistical analysis of the results with respect to the defined experimental factors provides further insights into the specific areas of challenge in code generation for each LLM. Our preliminary results on this modest benchmark demonstrate a framework for developing larger, comprehensive, unique benchmarks for both temporal financial tabular data and R code generation. While Python and SQL already have benchmarks, we are filling in the gaps for R. Powerful AI agents for text-to-code generation, as demonstrated in this work, provide a critical capability required for the next-generation AI-based natural language financial intelligence systems and chatbots, directly addressing the complex challenges presented by querying temporal tabular financial data.**

***Index Terms*—large language models, code generation, gpt-4, llama3, financial data, tabular data, benchmarking, financial industry, automation**

I. Introduction

In the finance sector, tabular datasets such as time-stamped transaction datasets are very common, and processing this data is crucial for making important decisions such as assessing risks of planning investments. This data can span a wide range of sources and vary greatly in structure. It can include details like market trends, transaction records, credit scores, etc. Handling these large, complex datasets can be difficult and slow since the exact information that needs to be extracted can require complex queries and processing to properly massage data into a usable form for the desired analyses. Fortunately, recent large-scale language models (LLMs) such as OpenAI's GPT models [1], [2] and Meta's LlaMA models [3] have demonstrated impressive abilities to generate code for requested tasks. Using LLMs, users can pose free-form questions that are then translated into code snippets in languages such as Python, SQL, or R and directly executed to extract the appropriate data and perform the required calculations. LLM agents that perform text-to-code translations for data retrieval, preparation, and analysis will be a critical component of the next-generation financial and business intelligence systems and chatbots [4]. Numerous benchmarks have demonstrated that LLMs can effectively generate Python and SQL [5]. However, how well LLMs can generate R code remains an open question. Also, as we demonstrate in this paper, dealing with queries about time requires special considerations to achieve good performance.

This project is driven by the overarching objective of meticulously examining and contrasting the code generation prowess of LLMs on temporal tabular data. Specifically, we evaluate the effectiveness of the state-of-the-art language models in generating code for querying tabular financial data in natural language using zero-shot prompting [6].If these models can consistently generate code that is both syntactically correct and generates the desired results, they would both

help analysts that query and process these kinds of data and achieve a necessary step in the process of developing AI-based natural language financial intelligence systems and chatbots. This work has the following contributions:

1) We demonstrate how to create zero-shot prompts for Python, SQL, and R to solve natural language queries about temporal financial tabular data. We see that special prompts are needed to correctly answer time-oriented questions.
2) We utilize experimental design to create a suite of queries to assess the performance of LLMs on natural language prompts on a large temporal financial dataset. We created a test data set consisting of 105 different queries with manually created R code answers. To investigate the strengths and weaknesses of LLMs, each query is created with different factors in mind that characterize the financial meaning of the queries and their complexity.
3) We execute this experiment on two LLMs and assess if the systems produced are executable and correct, and assess the efficiency of the produced code for Python, SQL, and R. We find that while LLMs have promising performance, challenges remain to achieve consistent performance across all three languages.
4) We perform a multi-factor analysis of the results with respect to the defined experimental factors to further understand the types of queries that are challenging for each LLM. This helps understand areas that must be improved to develop a fully reliable code generation system for temporal tabular transaction data.
5) We discuss the steps necessary to transform this promising preliminary work into full benchmarks of R code generation and generation for financial tasks, as such benchmarks currently do not exist to the best of our knowledge.

.

In this paper, we explore the effectiveness of different language models in generating code for performing queries of varying complexities on tabular financial datasets. In Section II, we describe our experiments for accomplishing this, such as the language models tested, the datasets for which queries were generated, the kinds of queries we sought to produce, etc. Section III is where we present our main results, showcasing the factors on which each model performs better or worse. In Section IV, we discuss these results and their importance. Then, we contextualize this work in Section V with related work, and finally, we talk about the next steps and conclusions in Section VI.

II. Methods and Experimental Design

At the heart of this endeavor lies a comprehensive exploration of how LLMs navigate the intricacies of coding syntax and semantics to produce coherent and effective code. The experimental design takes into account many key aspects. First and foremost, we want to experiment with multiple state-of-the-art language models to compare their capabilities. The models tested are OpenAI's GPT-4 [2] (specifically, gpt-4-0613) and Meta's Llama 3 [7] (specifically, Llama-3-70b-instruct). We also tested these models' abilities to generate code in different programming languages. We chose to generate and test code in three of the most popular languages for data processing: Python, R, and SQL, according to [8]. Having decided upon these high-level design choices, we designed a set of queries to be executed on financial datasets that span a variety of factors in order to create a comprehensive assessment of the performance of these models' capabilities. We describe the factors used in the query design and the datasets on which the queries are executed in Section II-A. Having designed our queries, we then evaluated the code that the language models generated. Our evaluation includes metrics such as the code complexity, syntactic correctness, and output correctness of the generated code. These are described in more detail in Section II-B.

A. Query and Prompt Design

1) Experimental Factors: The complexity present in financial datasets means that there is a large variety of operations and tasks that may need to be carried out in order to properly answer a question about a dataset. These operations vary in complexity and difficulty and often need to be strung together in order to reach the desired result. For these reasons, we designed natural language queries that vary across experimental factors designed to test the model's code generation capabilities in a wide variety of problems. These factors serve as evaluative criteria, guiding our analysis and comparison of code generation outcomes. Through systematic factor assignment and analysis, we aim to elucidate the impact of different query attributes on code generation performance. The resulting queries cover a wide range of tasks, including statistical analysis, data manipulation, transaction analysis, alias/user identification, data cleaning, and time analysis.

Table I provides the six query experimental factors developed to date. The first four factors characterize the operations involved in the code. These factors were created by human annotation of the R code. 'Filtering' includes operations such as `filter`, `selection`, and `pull`. 'Grouping' includes operations such as `group`, `summarize`, and `arrange`. 'Transformation' includes operations such as creating tables, `transform`, `mutation`, and `conversion`. 'Statistical analysis' includes operations such as `mean`, `maximum`, and `minimum`. 'Time' indicates that the query involves an analysis with respect to time.

For this preliminary analysis, we generated a total of 105 different queries with manually-generated solutions written in R. The factor 'Manual complexity' is the Halstead complexity [9] of the manually generated R solution. Thus, in our methodology, we used the Halstead complexity measurement as a robust technique for assessing the complexity of generated code snippets. These metrics are based solely on the counts of operators and operands in the code, regardless of the programming language used. By focusing solely on the counts of operators and operands in the code, Halstead measures abstract

away language-specific syntax and semantics, enabling fair and objective comparisons. We also evaluated the complexity of the manual solutions.

TABLE I
EXPERIMENTAL FACTORS CHARACTERIZING NATURAL LANGUAGE QUERIES WITH PERCENTAGE IN QUERIES.

Factor	Values (Percentage%)
Filtering	True (62), False (38)
Grouping	True (33), False (67)
Transformation	True (62), False (38)
Statistics	True (43), False (57)
Time	True (24), False (76)
Manual Complexity	Halstead Complexity of the R Solution

2) LLM System Prompts: System prompts were varied across each language to ensure the code was written in the correct format and with the desired language libraries. The system prompts for each language are provided in Table II. In each case, we ask the language models to output just the code with no surrounding text so that we can more easily execute the code automatically. For each language, we specifically instruct the models on how to convert the UNIX timestamps in the data into a more usable format. Without specific instruction, performance on date-related queries is much worse, as shown in Section III. Finally, in Python, we instruct the models to assign the final answer to a variable named "result" to extract the final answer from the returned code consistently.

3) User Prompts: The user prompts vary for each query and do not follow a structured template. Our intent was for this evaluation to test natural-language prompts. For any desired query, a user might word their query in many different ways. This is why we tested five different wordings for each query. For instance, one query was to find a list of users who have never been involved in liquidation transactions. Two of the ways this query was asked were "Give me all the users who have never liquidated or been liquidated" and "I need to see the list of users without any liquidation actions, neither active nor passive." In total, we tested 21 unique queries, each with five distinct wordings. This gives us 105 total queries tested. Once written, these queries are passed as user prompts with no further modification. The full list of queries is available on Github [1].

4) Temporal Tabular Financial Datasets: The underlying data on which the generated code is tested comes from Aave [10]. Aave is a decentralized lending protocol built on various blockchains that uses smart contracts to allow users to create "savings accounts" for their cryptocurrencies, earn interest on their deposited assets, and take out loans using these assets as collateral. For the purpose of this analysis, we used Aave's V2 [11] Mainnet deployment, which is on the Ethereum blockchain [12]. The transactions are posted on the Ethereum blockchain and represent real financial transactions.

[1]https://github.com/Large-Transaction-Models/Financial-Queries-Code-Generation

TABLE II
SYSTEM PROMPTS FOR LANGUAGE MODELS BASED ON PROGRAMMING LANGUAGE.

Language	System Prompt
R	You are an R programmer, following these rules: 1. When I ask you a question, give me the code to do it, without saying anything else. Do not put the code in an R chunk, just give me the code. 2. Use dplyr when possible to analyze data frames. Dplyr is already loaded, so do not use library(dplyr). 3. Never use install.packages. Assume every package you need is installed. 4. Always convert timestamps with as_datetime from the lubridate package before using them.
SQL	You are an SQL programmer, following these rules: 1. When I ask you a question, give me the code to do it, without saying anything else. Return the code as a single line. Do not put quotes around the code. 2. Always convert timestamps with the datetime(¡column¿, 'unixepoch') function before specifically using timestamps.
Python	You are a Python programmer, following these rules: 1. When I ask you a question, give me only the code to do it, without saying anything else. 2. Use pandas to analyze data frames. Pandas is imported as 'pd', and numpy is imported as 'np'. 3. Assign the answer to the question as a variable named 'result'. 4. Always convert timestamps with pd.datetime(unit = 's') before using them.

The data spans from January 1, 2021, through December 31, 2022, and includes 1,665,737 transactions made by 172,872 unique users. This dataset includes every transaction that was made in Aave V2 Mainnet over that time period, including deposits, withdrawals, borrows, repayments, liquidations, and collateral swap transactions. For each transaction, features such as the timestamp of the transaction, the user who made the transaction, the currency involved, the amount of currency involved, the adjusted USD value of the transaction, etc., are included. We also include queries for a secondary view of this data that was originally created for a paper about clustering user behaviors in Aave [13]. This view creates quarterly summaries of each user's behavior, including features such as how many transactions they performed in a quarter, the total USD value they borrowed in each quarter, the total USD value they deposited in each quarter, etc. We use these financial datasets and create natural language-based prompts that request certain information from each.

5) LLM Hyperparameters: To elicit code responses from the language models, we send the prompts in a structured format to the models through an API. There is a specific format and certain hyperparameters described in Table III that are used for querying each language model. These parameters include the model name for identification, minimum and maximum token requirements for input sequences, top-k and top-p values regulating token selection diversity and sampling probability, respectively, temperature affecting randomness, presence penalty penalizing specific token presence, and frequency penalty discouraging token repetition.

B. Code Evaluation Methodology

To evaluate and compare the code generated in the three languages, we use metrics capturing code complexity, code

TABLE III
HYPERPARAMETERS FOR QUERYING LANGUAGE MODELS: DEFAULTS USED EXCEPT FOR LLAMA 3'S MAX TOKENS INCREASED TO 4096 TO MATCH GPT-4.

Model Name	Min Tokens	Max Tokens	Top_k	Top_p	Temperature	Presence Penalty	Frequency Penalty
gpt-4-0613	0	4096	NA	1	1	0	0
Llama-3-70b-Instruct	0	4096	50	0.9	0.6	1.15	0.2

TABLE IV
RATE OF CORRECT CODE RESPONSES BY LANGUAGE, WITH STATISTICAL SIGNIFICANCE.

Model	Python	R	SQL	Total	χ^2 p-value
GPT-4-0613	0.952	0.790	0.876	0.873	0.002
Llama-3-70b-Instruct	0.581	0.714	0.657	0.651	0.127

correctness, and code execution. For every query, we manually wrote code that produces the desired result, so we know what the correct result should be. To determine correctness, we manually checked if the code responses generated by the language model returned the same information as the manually written code. Execution means that the code is syntactically correct, but does not necessarily mean that we get the correct result. Code in Python and SQL was converted to R-code for comparison with our hand-crafted benchmark. For Python, the `reticulate` package facilitated code execution and automatic conversion of outputs to R objects, while for SQL, the `sqldf` package was employed for similar conversion functionality.

III. RESULTS

A. Analysis by Language

The proportion of correct responses by model and language is given in Table IV. For GPT-4, Python was the best-performing language, followed by SQL and then R, while for Llama 3, the ordering was reversed: R was the best, followed by SQL and then Python. The column χ^2-square p-value indicates the p-value of a chi-square test for the rate of correct answers being equal for each language. The languages have significantly different correctness (p-value 0.002), but any observed differences are not significant (p-value 0.127) for Llama 3. GPT-4 performed better than Llama 3 for all languages.

TABLE V
RATE OF EXECUTABLE CODE RESPONSES BY LANGUAGE, WITH STATISTICAL SIGNIFICANCE.

Model	Python	R	SQL	Total	χ^2 p-value
GPT-4-0613	0.990	0.914	0.952	0.952	0.035
Llama-3-70b-Instruct	0.905	0.914	0.857	0.892	0.360

An analogous chart for comparing the rates at which the generated code is executable based on the programming language is shown in Table V and Figure 1. For GPT-4, Python had the highest rate of executable code, followed by SQL, then R. For Llama 3, R had the highest executable rate, followed by Python, then SQL. The difference between languages is significant with GPT-4 but not with Llama-3. Figure 1 displays summary information for result correctness and rates of executable code.

TABLE VI
CODE COMPLEXITIES OF MODEL RESPONSES BASED ON THE MODEL AND LANGUAGE.

Language	Mean_GPT	Mean_Llama	Std._GPT	Std._Llama
Python	36.476	26.868	14.587	14.314
R	20.471	17.288	8.555	5.993
SQL	24.333	30.190	13.170	17.966

Figure 2 shows a density plot of response complexity while Table VI shows summary statistics. R code tended to be the least complex in both models. Python was, on average, less complex than SQL when using Llama but more complex than SQL when using GPT.

B. Impact of Factors

We use logistic regression (LR) to determine the significance of the experimental factors in predicting the probability of correctness and execution of responses for each model. Table VII shows the results of an LR with the dependent variable being task correctness and independent factors being the prompt factors in Table I, and Table VIII shows the same with the dependent variable being code execution. `LanguageR` and `LanguageSQL` are dummy variables that are 1 if a response is in their respective language and 0 otherwise. Therefore, their coefficients are relative to the programming language Python.

The correctness of GPT-4 is negatively impacted by the use of the R and SQL languages (as compared to Python). GPT-4 shows a higher probability of correctness on prompts relating to filtering, transformation, and time, significant under a 5% level. In contrast, Llama 3 only shows a decrease in the probability of correctness for grouping prompts. GPT also shows decreased correctness probability for prompts with higher manual complexity. The LR supports that GPT-4 achieves less correctness in R and SQL than in Python.

For execution, GPT is significantly more likely to return executable code for prompts relating to filtering and transformation, and the code it returns is less likely to execute for higher manual complexities. Llama 3 once again shows a lower probability of executing for grouping queries.

IV. DISCUSSION

Our preliminary evaluation indicates that LLM code generation for temporal financial transaction data has great potential, but there is much room for improvement.

GPT-4 consistently outperforms Llama-3 in both correctness and execution rates across all three languages. This is

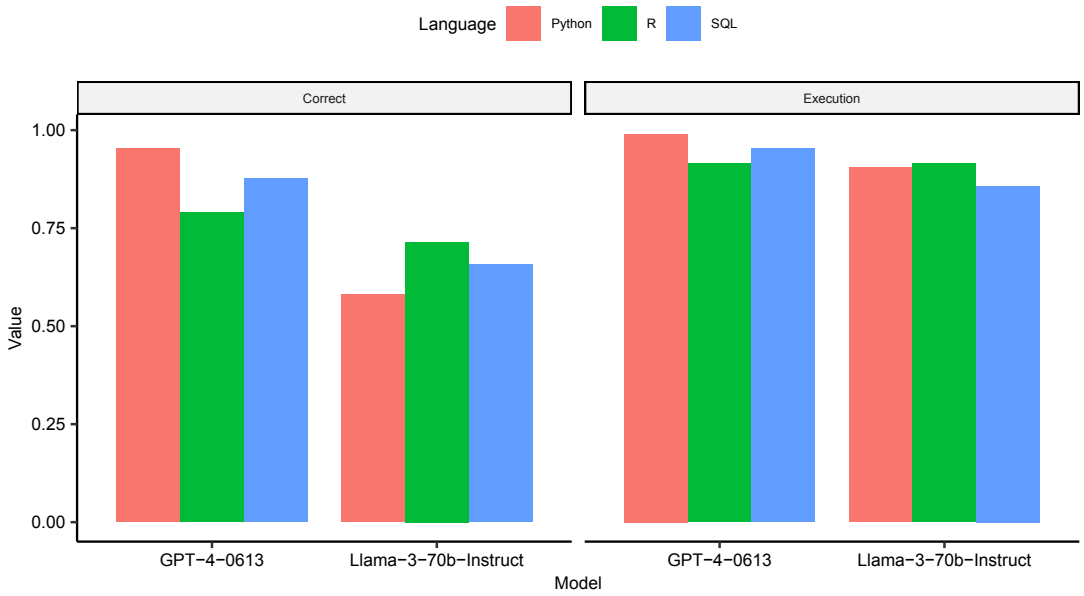

Fig. 1. Rate of Response Correctness and Execution.

TABLE VII
LOGISTIC REGRESSION FOR CORRECTNESS.

	Dependent variable:	
	Correct	
	GPT-4-0613	Llama-3-70b-Instruct
LanguageR	−1.879***	0.631**
	(0.546)	(0.303)
LanguageSQL	−1.138**	0.347
	(0.569)	(0.296)
Filtering	2.384***	0.481
	(0.711)	(0.387)
Grouping	0.909	−0.859**
	(0.632)	(0.347)
Transformation	2.968***	−0.038
	(0.682)	(0.316)
Statistics	1.565*	−1.091*
	(0.910)	(0.611)
Time	1.656**	0.805*
	(0.678)	(0.415)
Manual_Complexity	−0.144***	−0.025
	(0.045)	(0.026)
Intercept	2.159**	0.672
	(0.935)	(0.564)
Observations	315	315
Note:	*p<0.1; **p<0.05; ***p<0.01	

TABLE VIII
LOGISTIC REGRESSION FOR EXECUTION.

	Dependent variable:	
	Execution	
	GPT-4-0613	Llama-3-70b-Instruct
LanguageR	−2.500**	0.124
	(1.096)	(0.497)
LanguageSQL	−1.764	−0.498
	(1.132)	(0.452)
Filtering	2.949***	−0.236
	(1.033)	(0.650)
Grouping	2.140*	−1.866***
	(1.284)	(0.559)
Transformation	2.792**	−0.726
	(1.350)	(0.481)
Statistics	0.388	−1.052
	(1.214)	(1.193)
Time	1.661	−0.617
	(1.113)	(0.630)
Manual_Complexity	−0.283***	0.050
	(0.092)	(0.045)
Intercept	6.243***	3.038***
	(1.842)	(0.905)
Observations	315	315
Note:	*p<0.1; **p<0.05; ***p<0.01	

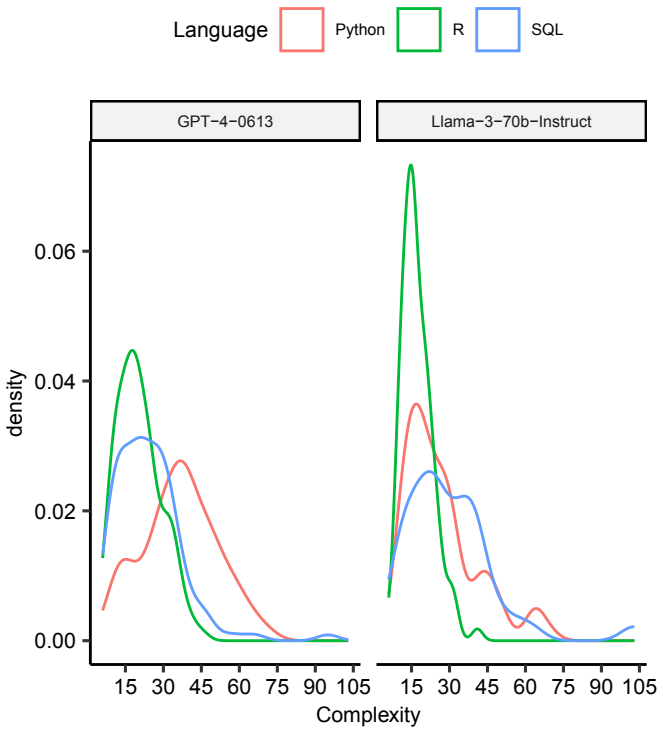

Fig. 2. Density Plot of Complexity.

evident from the higher rates of correct answers and execution for GPT-4 compared to Llama-3. By understanding which language model performs better for specific programming languages and tasks, developers can make informed decisions when selecting a model for their projects. This can lead to more accurate and efficient code generation, ultimately improving the quality of software products and reducing development time. GPT-4 exhibits significant language-specific performance variations, with Python being the best-performing language, followed by SQL and then R. In contrast, Llama-3 shows a reverse pattern, with R being the best-performing language, followed by SQL and then Python, but these differences are not statistically significant. This discrepancy suggests that the effectiveness of language models can vary depending on the programming language being generated. The discrepancy in performance across languages provides insights into the underlying capabilities of language models. GPT-4 is also better at generating executable code but still experiences some failures. This suggests that language models may have inherent biases or strengths that influence their effectiveness in generating code for different programming languages. By understanding these nuances, developers can better leverage language models to meet the specific requirements of their projects.

Financial data often has a temporal element to it, and this can take many different forms. Data such as stock prices may be recorded at regular intervals, e.g., daily, and recorded. User transactions within a financial platform will occur very irregularly and frequently and may be recorded to the nearest second or millisecond. Other data may be calculated quarterly or on some cycle. All of these methods of recording the temporal aspect of data result in different handling of the record of time. Through our testing, LLMs struggled severely at generating meaningful or correct queries when tasked with questions that forced them to deal with the time of the transactions in some way. For this reason, we had to add special instructions to the system prompt for each language that provided some basic details on how to manipulate the time variables in the code. Even with these special instructions, the models were not perfect when performing tasks that dealt with time.

The analysis of response complexity reveals interesting insights. R code tends to be the least complex across both models. Additionally, there's a discrepancy in the complexity of Python code between the two models. When using Llama-3, Python code is less complex on average compared to SQL. However, when employing GPT-4, Python code tends to be more complex than SQL. This suggests that the complexity of generated code can be influenced by both the language model and the specific programming language. Complex code can pose challenges for developers, increasing the time and effort required to comprehend and modify code. By understanding the factors that contribute to code complexity, developers can optimize their workflows and tooling to mitigate complexity-related issues, leading to improved productivity and faster iteration cycles.

LR analysis provides further insights into the factors influencing correctness and execution probabilities. Notably, GPT-4 shows a higher probability of correctness for prompts related to filtering, transformation, and time, while Llama-3 exhibits decreased correctness probability only for grouping prompts. Similarly, GPT-4 is more likely to return executing code for filtering and transformation prompts, while Llama-3 shows a lower probability of executing grouping queries. This highlights the models' strengths and weaknesses in handling different types of coding tasks. The significance levels reinforce the reliability of the observed patterns and highlight the practical implications of these findings for users of language models in code generation tasks.

The experimental design framework developed in the study is completely extensible and has the potential to be the basis of new benchmarks. The fact that we observed such variation in a relatively modest-sized test suite of prompts in terms of models, language, and query factors is quite encouraging. One limitation of the present study is its size. We are in the process of generating more natural language queries for all combinations of factors to provide a more comprehensive analysis. The other limitation is that we have just scratched the surface of possible financial queries. The framework can also be extended by both expanding the set of experimental factors that cover more aspects of financial transaction tasks and by asking these questions in different ways. We have also been examining adapting existing Python and SQL benchmarks to R code generation. A final limitation of the present study is that

we are currently evaluating correctness manually. We are in the process of developing automated methods for LLM-based agents to assess the correctness of the generated results. We believe that these strong early results show these are worthy endeavors.

V. Related Work

Beginning with the introduction of FinBERT [14], there has been a growing effort towards utilizing LLMs for various financial data analyses. FinTral [15] is a specialized LLM for the financial domain, integrating textual, numerical, tabular, and visual data processing. BloombergGPT [16] presents a 50-billion parameter model trained on extensive financial data, showcasing its prowess in financial tasks while maintaining robust performance in general LLM benchmarks. PIXIU [17] introduced a comprehensive framework featuring a financial LLM fine-tuned with instruction data, thereby advancing the open-source development of financial AI. Instruct-FinGPT [18] utilized instruction tuning to excel in scenarios requiring deep numerical understanding and contextual comprehension, particularly in financial sentiment analysis.

LLMs are extensively used on financial text data. LLMs have also been trained to deal specifically with financial document understanding, which can be used for tasks such as generating financial reports [19] and text summarization in a financial context [20]. GPT-InvestAR [21] aimed to enhance stock investment strategies by analyzing annual reports using LLMs. This approach yielded promising results in outperforming traditional market returns, highlighting the potential for LLMs in investment strategies. InvestLM [22] showed strong capabilities in understanding economic text and providing practical investment advice. Retrieval-augmented LLMs [23] addressed the challenges of applying LLMs directly to economic sentiment analysis, achieving considerable performance gains. [24] addressed the issue of hallucination in information extraction from earning call transcripts, achieving enhanced accuracy through the integration of retrieval-augmented generation techniques with metadata. FinLMEval [25] evaluated the performance of LLMs in financial natural language processing tasks, providing foundational assessments to guide ongoing improvements in LLMs within the financial domain. DISC-FinLLM [26] introduced a Chinese financial LLM using a Multiple Experts Fine-tuning Framework, demonstrating enhanced performance across various monetary scenarios compared to baseline models.

Benchmarks exist for Python and SQL code generation, but there is a notable lack of R benchmarks. For example, BIRD (BIg Bench for large-scale Database Grounded text-to-SQL Evaluation) [5] is a big benchmark for large-scale databases grounded in text-to-SQL parsing. The BIRD benchmark provides a challenging testbed for assessing the performance of SQL generation models in real-world scenarios. Recent thrust in Python code generation models also led to the development of several benchmark datasets. HumanEval [27] comprises 164 handwritten problems. The MBPP dataset [28] contains 974 entry-level problems. These benchmarks serve as standardized evaluation frameworks for assessing the performance of Python code generation models across a diverse range of tasks and challenges.

VI. Conclusion and Future Work

We have developed a preliminary framework for generating code to query temporal financial tabular data. Our experimental framework was used to evaluate the code generation capabilities of two prominent LLMs, GPT-4 and Llama 3, focusing on their efficiency and accuracy in handling complex queries and processing tabular financial data using R, Python, and SQL. Our findings indicate a clear superiority of GPT-4 in generating functionally correct and efficient code across all tested programming languages. Notably, GPT-4's performance in Python code generation was the best, achieving over 95% correctness on our testbed of prompts. We find that GPT-4 performance in R is considerably weaker at 79% correctness. However, the performance of the more compact Llama model on R at 71.4% is not far behind. For both models, the performance of their generated SQL code was not as strong as their strongest language (Python for GPT-4 and R for Llama 3), but better than their weakest language. However, for Llama 3, the generated SQL code was the least likely of the three languages to compile correctly. We note that we only employed zero-shot prompting using the same prompts for both LLM models. Results could be improved by employing few-shot prompting or fine-tuning and further developing the system prompts specifics for each LLM.

The experimental framework introduced in this research involved varying levels of complexity in the prompts, including the necessity to filter, group, transform, and compute statistical measures on temporal and non-temporal aspects of the data. These variations were developed for the rigorous assessment of the adaptability and accuracy of each model under diverse and challenging scenarios of financial interest. The superior performance of GPT-4 suggests that its training and underlying model architecture are better suited for tasks that require a deep understanding and manipulation of temporal tabular data, which is a common requirement in the financial sector.

Building on the insights gained from this study, we have several directions for future work. First, the total number and variety of tasks we ask the LLMs to perform is low, and the overall experiment would be improved by designing a larger-scale suite of tasks for evaluation. Second, expanding the scope of the experiments to include additional LLMs and other programming languages could provide a broader understanding of the general capabilities and limitations of current language models in code generation tasks. Varying the hyperparameters of each model beyond their defaults would also be an interesting addition to the experiment. Third, developing a standardized benchmark for evaluating LLM code generation abilities could catalyze research in this area. Such a benchmark should assess the correctness, efficiency, and complexity of the output code and also assess the models' performance on a wider variety of datasets. Such an expansion

of this work should also move us away from manual verification of the correctness of generated code and towards a more objective and efficient method. We can also investigate the usage of models that have been fine-tuned on finance-related tasks to see whether additional domain knowledge can impact the models' abilities to write effective code for finance-related data processing.

By pursuing these future directions, we aim to contribute to the evolving field of LLM applications in programming, particularly in enhancing their utility and reliability for professional use in the financial industry. This work will hopefully pave the way for more sophisticated, user-friendly, and robust LLM-based systems that can assume a larger role in financial data analysis and automation.

VII. Acknowledgments

We thank Arthi Seetharaman for her help with early versions of this work and Xiaoyang Liu for his advice on how to improve this paper. The authors acknowledge the support from NSF IUCRC CRAFT center research grants (CRAFT Grants #22003, #22006) for this research. The opinions expressed in this publication and its accompanying code base do not necessarily represent the views of NSF IUCRC CRAFT. We also would like to thank Amberdata for providing a portion of the data used in this work.

References

[1] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49313245

[2] OpenAI, "Gpt-4 technical report," 2024.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023.

[4] M. Azmi, A. Mansour, and C. Azmi, "A context-aware empowering business with ai: Case of chatbots in business intelligence systems," *Procedia Computer Science*, vol. 224, pp. 479–484, 2023.

[5] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Cao, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. C. C. Chang, F. Huang, R. Cheng, and Y. Li, "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls," 2023.

[6] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[7] Meta AI, "LLaMA3," https://llama.meta.com/llama3/, accessed: Apr 25, 2024.

[8] edX, "Nine Top Programming Languages for Data Science," jan 2024, available online. [Online]. Available: https://www.edx.org/resources/9-top-programming-languages-for-data-science

[9] B. Curtis, S. B. Sheppard, P. Milliman, M. Borst, and T. Love, "Measuring the psychological complexity of software maintenance tasks with the halstead and mccabe metrics," *IEEE Transactions on software engineering*, no. 2, pp. 96–104, 1979.

[10] Boado, Ernesto, "AAVE Protocol Whitepaper," Tech. Rep., 01 2020. [Online]. Available: https://www.cryptocompare.com/media/38553941/aave_protocol_whitepaper_v1_0.pdf

[11] ——, "AAVE Protocol Whitepaper V2.0," Tech. Rep., 12 2020. [Online]. Available: https://cryptorating.eu/whitepapers/Aave/aave-v2-whitepaper.pdf

[12] V. Buterin, *Ethereum: A next-generation smart contract and decentralized application platform*, 2014. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[13] A. Green, M. Giannattasio, K. Wang, J. S. Erickson, Oshani, Seneviratne, and K. P. Bennett, "Characterizing common quarterly behaviors in defi lending protocols," 2023. [Online]. Available: https://www.marble-conference.org/marble2023-cfp

[14] D. Araci, "Finbert: Financial sentiment analysis with pre-trained language models," 2019.

[15] G. Bhatia, E. M. B. Nagoudi, H. Cavusoglu, and M. Abdul-Mageed, "Fintral: A family of gpt-4 level multimodal financial large language models," 2024.

[16] S. Wu, O. Irsoy, S. Lu, V. Dabravolski, M. Dredze, S. Gehrmann, P. Kambadur, D. Rosenberg, and G. Mann, "Bloomberggpt: A large language model for finance," 2023.

[17] Q. Xie, W. Han, X. Zhang, Y. Lai, M. Peng, A. Lopez-Lira, and J. Huang, "Pixiu: A large language model, instruction data and evaluation benchmark for finance," 2023.

[18] B. Zhang, H. Yang, and X.-Y. Liu, "Instruct-fingpt: Financial sentiment analysis by instruction tuning of general-purpose large language models," 2023.

[19] C. L. Chapman *et al.*, "Towards generating financial reports from tabular data using transformers," in *Machine Learning and Knowledge Extraction*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 221–232. [Online]. Available: https://doi.org/10.1007/978-3-031-14463-9_14

[20] M. La Quatra and L. Cagliero, "End-to-end training for financial report summarization," in *Proceedings of the 1st Joint Workshop on Financial Narrative Processing and MultiLing Financial Summarisation*, D. M. El-Haj, D. V. Athanasakou, D. S. Ferradans, D. C. Salzedo, D. A. Elhag, D. H. Bouamor, D. M. Litvak, D. P. Rayson, D. G. Giannakopoulos, and N. Pittaras, Eds. Barcelona, Spain (Online): COLING, Dec. 2020, pp. 118–123. [Online]. Available: https://aclanthology.org/2020.fnp-1.20

[21] U. Gupta, "Gpt-investar: Enhancing stock investment strategies through annual report analysis with large language models," 2023.

[22] Y. Yang, Y. Tang, and K. Y. Tam, "Investlm: A large language model for investment using financial domain instruction tuning," 2023.

[23] Y. Zhang, Y. Yang, B. Liang, S. Chen, B. Qin, and R. Xu, "An empirical study of sentiment-enhanced pre-training for aspect-based sentiment analysis," in *Findings of the Association for Computational Linguistics: ACL 2023*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 9633–9651. [Online]. Available: https://aclanthology.org/2023.findings-acl.612

[24] B. Sarmah, T. Zhu, D. Mehta, and S. Pasquali, "Towards reducing hallucination in extracting information from financial reports using large language models," 2023.

[25] Y. Guo, Z. Xu, and Y. Yang, "Is chatgpt a financial expert? evaluating language models on financial natural language processing," 2023.

[26] W. Chen, Q. Wang, Z. Long, X. Zhang, Z. Lu, B. Li, S. Wang, J. Xu, X. Bai, X. Huang, and Z. Wei, "Disc-finllm: A chinese financial large language model based on multiple experts fine-tuning," 2023.

[27] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," 2021.

[28] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton, "Program synthesis with large language models," 2021.