Evaluating Financial Literacy of Large Language Models through Domain Specific Languages for Plain Text Accounting

Alexei Figueroa[1], Paul Grundmann[1], Julius Freidank[1,2], Alexander Löser[1], Wolfgang Nejdl[3],
[1]Berlin University of Applied Sciences (BHT)
[2]Humboldt University Berlin
[3]Leibniz University Hannover
Correspondence: afigueroa@bht-berlin.de

Abstract

Large language models (LLMs) have proven highly effective for a wide range of tasks, including code generation. Recently, advancements in their capabilities have shown promise in areas like mathematical reasoning, chain-of-thought processes and self-reflection. However, their effectiveness in domains requiring nuanced understanding of financial contexts, such as accounting, remains unclear. In this study, we evaluate how well LLMs perform in generating code for domain-specific languages (DSLs) in accounting, using `Beancount` as a case study. We create a set of tasks based on common financial ratios, to evaluate the numeracy and financial literacy of LLMs. Our findings reveal that while LLMs are state-of-the art in generative tasks, they struggle severely with accounting, often producing inaccurate calculations and misinterpreting financial scenarios. We characterize these shortcomings through a comprehensive evaluation, shedding light on the limitations of LLMs in understanding and handling money-related tasks.

1 Introduction

In recent years, natural language processing methods and transformer models have seen significant improvements in text, language and coding related tasks. Especially the release of the pre-trained large language model GPT-3 (Brown et al., 2020) and its derivative ChatGPT (Schulman et al., 2022) to the general public have generated considerable public interest. Large language models (LLMs) have the ability to understand and generate text in a wide spectrum of disciplines and tasks, and are also able to generate code. These properties are leveraged across several industries to automate e.g., customer support and content creation. Although these models have had a significant impact, they are not sufficiently studied in the accounting practice, despite their potential to enable process automation.

Accounting in digital business practice. In contemporary business, enterprise resource planning (ERP) systems are a commonplace phenomenon, providing the foundation for a multitude of organisational functions and decision-making processes. ERP software typically supports a plethora of business dimensions, including human resource management or supply chain management among others. These systems provide a centralized platform for managing operations, offering features like analytics and automation to improve production and decision-making. Accounting plays a pivotal role in the functioning of ERPs, serving as a fundamental pillar upon which various business segments are constructed. It is responsible for ensuring the accuracy of financial data and the monitoring of budgetary allocations throughout the whole economic activity of a company. However, ERP systems can be rigid and complex, requiring human training and often leading to a bottleneck in user interaction, hindering efficiency and accessibility. In this work we aim to evaluate whether open-weight LLMs can accurately and efficiently perform accounting tasks using plain text accounting domain specific languages (DSLs). We investigate their ability to understand financial ratios, by generating accounting scenarios that affect them, e.g., selling a property to increase the Current Ratio (CuR).

LLMs for plain text accounting. We find in the DSLs of plain text accounting (PTA) the ideal target for LLMs to interface with financial transactional data. PTA is an accounting paradigm to record transactions in a human readable format with DSLs like `Ledger` (Wiegley, 2023), `hledger` (Michael, 2023) or `Beancount` (Blais, 2023). These languages are strictly compiled and incorporate double-entry accounting principles that can partially categorize the error classes of transactions generated by LLMs. We create two tasks for LLMs to generate scenarios motivated by the

Proceedings of the Joint Workshop of the 9th FinNLP, the 6th FNP, and the 1st LLMFinLegal, pages 63–75
January 19–20, 2025.

semantics of common financial ratios (e.g., Current Ratio). These quantities are generally used by financial practitioners to assess the economic state of a company and are a good proxy for financial literacy. Furthermore, with these scenarios, we also explore the capabilities of LLMs to generate corresponding transactions using a DSL, which we evaluate with a compiler. We subsample and thoroughly examine the results of the generation with the help of experts in the field. In our evaluation LLMs generally show significant problems regarding financial literacy and numeracy by extension. We characterize these essential deficiencies through six financial scenario error classes, and six transaction error classes. Our contributions can be summarized as follows:

- To our knowledge, we present the first analysis of the performance of LLMs in financial transaction generation.
- We create two novel tasks probing financial literacy of LLMs motivated by financial ratios.
- We provide an in-depth error analysis on LLM powered plain text accounting generation with 12 error classes among two different tasks.

We provide all data to replicate our experiments including our prompts, data and methodology. [1]

2 Related Work

Language models for code generation. In addition to common language-related tasks, LLMs are also applied in the field of code generation. There are three types of transformer models: encoder-only, encoder-decoder, and decoder-only. Code2Vec (Alon et al., 2019) is one of the first language models to attempt to understand code by representing code snippets as embeddings. Encoder-only models include CodeBERT (Feng et al., 2020) and CuBERT (Kanade et al., 2020), which are pretrained BERT models (Devlin et al., 2019) and are typically utilized in search or classification tasks. Encoder-decoder models, such as AlphaCode (Li et al., 2022) and CodeT5+ (Wang et al., 2023), are instrumental for source code summarization, text-to-code and code editing (Wang et al., 2021; Ahmad et al., 2021). Recently, decoder-only Transformer models, such as Codex (Chen et al., 2021), CodeGeeX (Zheng et al., 2023b), StarCoder (Li et al., 2023) and Wizardcoder (Luo et al., 2024), comprehend the state-of-the-art in generating code from natural language descriptions. In our work, we use LLMs as generators of accounting DSLs that can be inherently evaluated via compilation.

Large language models on accounting tasks. LLMs are leveraged to perform accounting tasks, such as auditing (Eulerich and Wood, 2023; Gu et al., 2023; Emett et al., 2023; Li and Vasarhelyi, 2023) and analyzing financial statements (Kim et al., 2024). Eulerich and Wood (2023); Emett et al. (2023) show that ChatGPT can help in open generation tasks such as audit report writing. Kim et al. (2024) examine the ability of LLMs (namely GPT-4) to analyze financial statements. Their findings suggest that GPT-4 and human analysts complement each other and chain-of-thought prompting (CoT) (Wei et al., 2024) leads to significantly better results. Gu et al. (2023) also make use of CoT prompting for co-piloted auditing and present financial ratio analysis, post-implementation review and journal entry testing as examples. We leverage CoT as a framework to direct and enhance the output of LLMs for our accounting scenario tasks.

Plain text accounting tools. In contrast to common ERP systems, plain text accounting is based on human-readable text files, which facilitates access and editing of transactions by both humans and machines. Among the most popular tools for plain text accounting are `Ledger` (Wiegley, 2023), `hledger` (Michael, 2023) and `Beancount` (Blais, 2023). `Beancount` offers features that are tailored to domains such as trading and investing and provides the most customization. The corresponding compiler is highly *pessimistic* and follows a strict approach assuming an unreliable user. Hence, we use `Beancount` as a target for the evaluation of our experiments.

3 Tasks and Dataset

3.1 Financial Ratios

Financial ratios cover all scopes of the situation of a company including operational applications in single departments, the entire company, or even external stakeholders such as suppliers and customers (Bragg, 2012, p. 1). The financial ratios that we consider are the liquidity ratios that ascertain a company's viability for investors. A company can be deemed to be viable when it maintains an amount of liquid assets that is sufficient enough to

[1] https://github.com/DATEXIS/LLMFinLiteracy

```
SCENARIO 1
Description: The company sells a property
for 500 EUR to increase liquidity
Effect: Positive
-----------------------------------------
TRANSACTION 1
2024-07-11 * "Selling non current asset"
  Assets:NonCurrent:Appartment -500 EUR
  Assets:Current:Cash           500 EUR
```

Figure 1: Top: Expected generated output of scenario generation increasing the Current Ratio (liquidity). Bottom: expected generated transaction using the Beancount DSL altering the balance-sheet accordingly.

pay off short-term liabilities (Bragg, 2012, p. 67). We focus on three liquidity ratios: Current Ratio, Quick Ratio and the Cash Ratio. These are among the most common liquidity ratios and require only accounts belonging to the balance sheet.

Current ratio. The current ratio assesses a company's capacity to pay short-term debt that matures within a year. The minimum level of liquidity is often considered to be at a current ratio of 1:1, where ratios closer to 2:1 are more desirable (Bragg, 2012, p. 81).

$$\text{Current Ratio} = \frac{\text{Current Assets}}{\text{Current Liabilities}} \quad (1)$$

Quick ratio. As the current ratio includes inventory which could overestimate the measured liquidity. The quick ratio alleviates this by excluding inventory when aggregating current assets. This results in a more balanced quantity that reflects how *quickly* accessible assets can be converted into cash (Bragg, 2012, p. 82).

$$\text{Quick Ratio} = \frac{\begin{array}{c}\text{Cash} \\ + \text{Marketable Securities} \\ + \text{Accounts receivable}\end{array}}{\text{Current Liabilities}} \quad (2)$$

Cash ratio. This ratio only considers how cash and cash equivalents can cover short-term liabilities. Since the cash ratio does not include assets that have to be transferred to cash, it is a direct and reliable indicator of liquidity (Bragg, 2012, p. 83).

$$\text{Cash Ratio} = \frac{\text{Cash} + \text{Cash Equivalents}}{\text{Current Liabilities}} \quad (3)$$

3.2 Tasks

We use LLMs to perform two primary tasks: *Generation of financial scenarios* and *Generation of transactions*. By generating financial scenarios, we assess whether LLMs are literate regarding accounting concepts, e.g., financial ratios and resource allocation within a company. Based on these scenarios, we generate transactions in the `Beancount` DSL. Using the respective compiler, we gauge whether LLMs understand double-entry accounting and can keep the context of an entire balance sheet as humans do. Additionally, we probe for numeracy w.r.t monetary quantities. Both tasks represent skills that are fundamental to the activities of financial practitioners.

Scenario generation. In this task LLMs generate scenarios that strategically influence financial ratios in the context of a balance sheet, specifically liquidity ratios. The expected output consists of a series of textual scenario descriptions and their *positive* or *negative* effects on a given liquidity ratio. An example of such scenario is presented in Figure 1 (top).

Plain text DSL transaction generation. In this task LLMs must translate the previously generated scenarios into plain text transactions, specifically those that are compilable by `Beancount`. This task assesses the models' ability to convert theoretical changes in financial ratios into practical accounting entries. These entries can be compiled by `Beancount` and be automatically categorized in different error classes. An example of an expected transaction is shown in Figure 1 (bottom).

Transactions are generally additive towards a balance sheet (initial state). However, the resulting changes in financial ratios are subject to the initial conditions as well as the arithmetic on the accounts. Thus, the changes on the balance sheet compared against the scenario objective effectively probe for numeracy and the intuition of arithmetics in LLMs.

3.3 Balance Sheet Data

We use real balance sheet statements to provide LLMs with the initial state (context) for the generation tasks. We focus on balance sheets of five different companies that are part of the DAX and have varying fields of operation: Airbus, Bayer, Deutsche Telekom, Mercedes and SAP, specifically their quarterly reports (Airbus, 2024; Bayer, 2024; Deutsche Telekom, 2024; Mercedes-Benz Group, 2024; SAP, 2024).

To ensure uniform naming of the accounts, the balance sheets are converted into the `Beancount` DSL. These are then used as part of the context

included in the prompt for the LLMs to process. An example of this company data expressed in the DSL is presented in Appendix C.

4 Experiments

4.1 Large Language Models

We include five state-of-the-art LLMs in our evaluation. We focus on smaller open weight models that can be deployed on premise following the privacy sensitivity of financial data. Our interest lays in discriminating between the performance of specialized code models and general purpose models. Hence, we evaluate three general purpose models: Llama-3-8B-Instruct (AI@Meta, 2024), Qwen-2-7B-Instruct (Yang et al., 2024), and Mistral-7B-v0.3 (Jiang et al., 2023), in addition to two models with a focus on coding: CodeLlama-7b-Instruct-hf (AI@Meta, 2023) and CodeQwen1.5-7B-Chat (Bai et al., 2023).

We limit the maximum number of generated tokens to 8192 per example and use greedy sampling with a temperature setting of 0.

4.2 Prompt Engineering

To provide the various models with the context of their tasks we use a standardized prompt protocol. We follow the principles of (Gu et al., 2023) and adapt their chain of thought (CoT) structure to the novel tasks. In total, the CoT prompts consist of nine prompts that guide the models in performing their tasks. The chain starts with a role definition and is then followed by a task explanation, an input data explanation, output data explanation, the plain text accounting rules, an input-output example, the balance sheet context, and the two task execution prompts. We provide details on the prompt protocol in Appendix A. We evaluate the LLMs' inherent domain knowledge regarding financial ratios, thus we do not provide explicit formulas.

4.3 Double-entry Accounting and the Beancount DSL

We use this DSL as the target of the *transaction generation* task since it can be compiled (see Appendix B for syntax details). The `Beancount` compiler validates that the postings follow a double-entry bookkeeping approach which is an industry standard. In double-entry bookkeeping, when an account is credited by an amount, a different account (or set of accounts) has to be debited by a corresponding inverse amount. The overall sum of all amounts of the transaction must be zero. In order to prevent errors in the accounts, the `Beancount` compiler verifies that the total of all postings across all transactions is zero. If the accounts do not balance to zero after the transactions, the `Beancount` compiler returns an error. All accounts in `Beancount` are categorised into one of five groups: Assets, Liabilities, Income, Expenses and Equity, where Equity is a summary of Income and Expenses (Blais, 2023). Since the scenarios generated focus on liquidity ratios, income and expenses are excluded as they do not affect the ratios directly.

4.4 Evaluation Setup

A domain knowledge expert evaluates the generated responses by each model in relation to the financial goal and outcomes of every task. For every model, financial ratio and company a model generates a response, resulting in a total of 1500 samples. Each sample contains a scenario and a set of corresponding financial transactions. To expedite the human evaluation process, we sub-sample this resulting dataset. We sample 60 entries for each of the five models, stratified (Parsons, 2017) by the combinations of company, scenario, and financial ratio. This results in a total of 300 data entries.

Human evaluation of scenarios. We follow a hierarchical approach, starting with the identification of major problems, such as missing scenarios, and ending with the evaluation of finer details, such as the correctness of the scenario content. As soon as an error occurs, the evaluation is stopped and no further reviews are carried out for the scenario. The error classes are evaluated in the following order:

1. **Missing Scenario**: a scenario is missing.
2. **Missing Effect**: the effect is missing.
3. **Ambiguous Accounts**: the affected accounts are not specific to the financial ratio.
4. **Scenario Incorrect**: the scenario content deviates from standard business practice (e.g., selling your own debt for cash).
5. **Effects Incorrect**: the effects of the scenario is inconsistent with the financial ratio.
6. **Correct**: the scenario-effect combination meets all criteria.

Evaluation of transactions. We distinguish between six error classes for the evaluation of transactions:

1. **Missing Transaction**: a transaction was not generated.
2. **Syntax Error**: the transaction format is incorrect.
3. **Unknown Account**: the account is not in the balance sheet.
4. **Balance Error**: the transaction does not balance to zero.
5. **Incorrect | Compiles**: the transaction compiles, but does not match scenario.
6. **Correct | Compiles**: the transaction compiles and the content is valid.

We append every transaction generated by the LLMs to to the corresponding company ledger. Then, we compile the resulting `Beancount` file. The resulting error messages are mapped to the respective transaction error classes. In cases where the compiler reports both balance errors and unknown account errors simultaneously, we prioritize account errors, since resulting balances are undefined. While a non-compiling transaction serves as a definitive indicator of an error, a compiling transaction does not necessarily signify correctness. Since the transactions are based on generated scenarios, they may not always accurately reflect the scenario. Therefore, we manually verify all the transactions that are compiled, checking that they are coherent with the scenario (*Incorrect | Compiles* and *Correct | Compiles*).

5 Evaluation Results

Human evaluation of scenarios. We report the distribution of the scenario classes across the 300 samples in Table 1. The distribution of the error classes reveals significant issues. 33.67% of all generated scenarios is missing and 6.33% are not describing any effect. Additionally, 28.33% of the scenarios included ambiguous accounts, making a clear assessment impossible. 14.33% of the scenario descriptions fail to adhere to accounting principles i.e., they are nonsensical. Furthermore, 5.67% of the scenarios are sufficiently specified, but do not affect the respective ratio as stated. Only 11.67% of the generated scenarios can be considered correct, following standard accounting practices and are coherent with their respective ratios.

Scenario Class	Proportion [%]
Missing Scenario	33.67
Missing Effect	6.33
Ambiguous Accounts	28.33
Incorrect Scenario	14.33
Incorrect Effect	5.67
Correct	11.67

Table 1: Distribution of Scenario Classes in %. A majority of the scenarios have missing or unspecified elements, highlighting significant gaps in completeness and accuracy.

General purpose models outperform. Table 2 details the performance across the different language models. Among these, only CodeLlama and CodeQwen 1.5 have missing scenarios. In fact, CodeQwen 1.5 fails to generate any scenario, while the outputs of CodeLlama lack the effect in 31.67% of the cases. In contrast, Mistral, Llama 3, and Qwen 2 do not have any missing scenarios nor effects, demonstrating a better adherence to the desired output structure. However, Mistral and Llama 3 struggle with specifying affected accounts in their scenarios, where 58.33% and 51.67% of scenarios exhibit this error, respectively. Qwen 2 stands out with the highest correct scenario generation accuracy of 21.67%. Mistral follows with an accuracy of 20%, while Llama 3 achieves an accuracy of 16.67%. Although, general purpose models outperform the code-related variants, the overall performance leaves a great room for improvement.

Transactions. We report the distribution of the transaction error classes across the 300 samples in Table 3. From these entries, 40% are missing the associated transaction. Additionally, 23.33% of the transactions do not balance, which suggests inconsistencies in the associated amounts, e.g., sign errors or mismatches in values. Furthermore, 17.67% of the transactions reference an unknown account and only 19% of the transactions adhere to the `Beancount` syntax. However, more than half of these (10.67% of the total) are nonsensical or inconsistent with the described scenario. Out of all evaluated transactions only 8.33% are correct. We expand on the performance of each model in Table 4.

Scenario Class	CodeLlama	CodeQwen 1.5	Mistral	Llama 3	Qwen 2
Missing Scenario	68.33	100.00	0.00	0.00	0.00
Missing Effect	31.67	0.00	0.00	0.00	0.00
Ambiguous Accounts	0.00	0.00	58.33	51.67	31.67
Incorrect Scenario	0.00	0.00	21.67	23.33	26.67
Incorrect Effect	0.00	0.00	0.00	8.33	20.00
Correct	0.00	0.00	20.00	16.67	21.67

Table 2: Distribution of Scenario Classes Across Models in %. CodeLlama and CodeQwen 1.5 fail to generate any correct scenarios. Mistral, Llama 3, and Qwen 2 show higher, though still suboptimal, accuracy, with Qwen 2 performing best.

Transaction Class	Proportion [%]
Missing Transaction	40.00
Syntax Error	0.00
Unknown Account	17.67
Balance Error	23.33
Incorrect | Compiles	10.67
Correct | Compiles	8.33

Table 3: Distribution of Transaction Classes in % The majority of generated transactions are not compiled or are flawed. Only 8.33% of all transactions are correct and compile.

CodeLlama and CodeQwen 1.5 fail to generate any transactions, which is expected considering their poor performance on generating scenarios. In contrast, Mistral, Llama 3 and Qwen 2 successfully generate transactions, albeit with varying error rates. The Qwen 2 model mainly generates transactions that do not balance (61.67%). Of the transactions generated by Mistral, 28.33% compile successfully but show inconsistencies with the scenarios they are intended to represent. This class is less pronounced in the transactions generated by Llama 3 (11.67%) and Qwen 2 (16.67%). The model that exhibits the best performance is Qwen 2, with 16.67% of transactions being compiled and correct. Llama 3 and Mistral achieve an accuracy of 15% and 10% respectively. Overall, Qwen 2 shows the highest accuracy, but generally, all models demonstrate significant limitations in generating correct transactions.

6 Discussion

Task generalization from context. We observe significant limitations in the ability of the chosen language models to generate accurate financial scenarios and transactions. The LLMs that are specialized in code generation performed significantly worse than the general models. In fact, they do not generate a single correct scenario. We argue that this is due to a high sensitivity of the model response to the prompt structure. These models consistently produce the string `"Processed - Waiting for next input."` after receiving the task prompt, resulting in no viable scenarios or transactions being generated. While this problem could potentially be mitigated with different prompting strategies, we do not explore this further and leave it as future work.

Among the scenarios generated by the general purpose models, nearly half are incomplete, often due to unspecific account descriptions. Only Qwen 2 generates scenarios with an error rate in the accounts of less than 50%. This result is problematic because accounting is typically a field in which accuracy is of paramount importance.

Transaction accuracy and financial literacy. Even though Mistral, Llama 3, and Qwen 2 do not generate any transactions with syntax errors, thus capturing the `Beancount` DSL, they compile less than 40% of the time, with Llama 3 having the lowest compile rate of 26.67%. This emphasizes how these models are not able to distinctively capture the accounts nor amounts in the scenario context. Furthermore, many of the transactions that did compile were inconsistent with the financial scenario (*Incorrect | Compiles*), highlighting the necessity of manual evaluation to avoid errors if these models were deployed in practice. We show an example in fig. 2, here the scenario is supposed to affect positively the Current Ratio (CuR), i.e. either increase liquid assets (like cash) or decrease short term liabilities. Although the transaction balances correctly, the transaction is nonsensical, since it describes the sales of debt while, in accounting terms, what it effectively accomplishes is to increase cash by increasing debt. This is a clear sign of the model

Transaction Class	CodeLlama	CodeQwen 1.5	Mistral	Llama 3	Qwen 2
Missing Transaction	100.00	100.00	0.00	0.00	0.00
Syntax Error	0.00	0.00	0.00	0.00	0.00
Unknown Account	0.00	0.00	45.00	35.00	8.33
Balance Error	0.00	0.00	16.67	38.33	61.67
Incorrect | Compiles	0.00	0.00	28.33	11.67	13.33
Correct | Compiles	0.00	0.00	10.00	15.00	16.67

Table 4: Distribution of Transaction Classes Across Models in %. CodeLlama and CodeQwen 1.5 fail to generate correct transactions. Transactions generated by Llama 3 and Qwen 2 mainly suffer from balance errors and Mistral from unknown accounts. Qwen 2 outperforms the other general-purpose models slightly.

```
Llama-CuR-Deutsche_Telekom13:
The company receives 700 EUR in cash from the sale of non-current financial liabilities.
Effect: Positive

2024-08-15 * "Sale of Non-Current Financial Liabilities"
  Liabilities:NonCurrent:FinancialLiabilities    -700 EUR
  Assets:Current:CashAndCashEquivalents           700 EUR
```

Figure 2: Succesfully compiling transaction created by the Llama 3 model for the scenario targeting an increase of the Current Ratio (CuR). Although the transaction balances correctly (zero sum), it is incoherent with the scenario. More importantly it's description and intent (sales of liabilities) expressed with these two accounts are nonsensical and show a clear hallucination regime.

following the syntax of the DSL, but hallucinating w.r.t the actual goal of the task.

Probing GPT-4o as a judge. We probe, whether an LLM-as-a-judge (Zheng et al., 2023a) for evaluation is feasible using the current state-of-the-art LLM GPT-4o. We use ten of the nonsensical scenarios yielded by our models as an input. We examine the output of GPT-4o with the help of a domain expert. The evaluation shows that GPT-4o fails to assess the underlying inconsistencies in all tested cases. This implies that even the current state-of-the-art can not be used for an automatic evaluation, highlighting the importance of human evaluation even for trivial accounting tasks.

General accounting performance. Generally, only 7 out of 300 (2.3%) scenario-transaction combinations resulted in a correct outcome. When excluding the code models, the accuracy only increases to 3.8%. For these correct samples, the generated scenario-transaction combinations resemble the provided examples in the context very closely. This suggests a possible over-reliance on the examples provided in the prompts, rather than demonstrating an ability to generalize or generating original results. Such behavior is potentially problematic, as it suggests that the models may be reproducing the patterns in the example scenarios rather than understanding the underlying processes or principles required for accounting.

Value proposition of LLMs for accounting. Given the significant time and compute required to generate even the seven correct scenario-transaction combinations, it is questionable whether LLMs are suited for generating plain text accounting files. The slow inference and low accuracy raise concerns about their efficiency and reliability in these tasks. Our human evaluation took approximately six expert hours to yield seven correct transactions, which represent only two financial ratios. This effectively reduces the number of valid scenario-transactions to two, which in reality would be significantly less time-consuming for a human practitioner.

Another critical factor is that even when LLMs manage to generate compiling transactions, the results can often be incorrect. This directly implies that it is impossible to use these technologies without human interaction. Transactions that the model compiles still require meticulous review by a qualified accountant to ensure that there are no content errors.

In essence, the lack of accuracy and the need for extensive post-processing review raises significant questions about the potential value of using LLMs

to automate accounting processes.

7 Conclusion

We evaluate the capabilities of open-weight large language models in generating meaningful accounting scenarios and code for plain text accounting with domain specific languages. Through a comprehensive evaluation of two novel tasks we gauge the domain knowledge and financial numeracy of these models. We highlight that the models show very poor performance. In a human expert evaluation we find that only for 2.3% scenario-transaction generations, LLMs succeed at our tasks, with most of them stemming from a single model (Llama 3).

These results raise significant concerns about the practical applicability of LLMs for code generation using domain-specific languages for accounting. Our results show that even successfully compiled (balanced) transactions can be flawed (e.g. hallucinated effects), severely propagating errors in an automated evaluation and assessment of results. Although we evaluate state-of-the-art prompt engineering techniques, these seem to be limited towards steering LLMs to a useful generation of transactions. This is worsened by the time-intensive nature of both inference and scenario human evaluation, which further complicates the search for a "*golden prompt*".

7.1 Future Work

Prompt engineering. Given that a significant proportion of the code model output was incomplete or missing, further refinement of the prompts and additional strategies could improve performance. We limit our survey to a Chain of Thought approach, and although it is state-of-the art, additional methods and experiments could be considered.

Hyperparameter optimization. A qualitative flaw of the generated scenarios is that they lack in originality (diversity). This potentially stems from the temperature we set to 0. A temperature of 0 results in greedy decoding, where the model selects the token with the highest probability at each step, leading to deterministic outcomes. By experimenting with different hyperparameters, such as using a temperature above zero or using a different search algorithm (e.g. beam search (Freitag and Al-Onaizan, 2017)), we can potentially get more diverse and original results.

Fine-Tuning on domain-specific datasets. Another potential area for improvement is fine-tuning the LLMs. The used general-purpose models were trained on very diverse corpora, which likely do not include sufficient data on accounting practices and `Beancount`. By fine-tuning the language models on domain-specific datasets, such as financial reports, accounting scenarios and `Beancount` files, the performance could be improved. Using a more specialized dataset, the models could learn to generate scenarios and transactions that are not only syntactically correct, but also align more closely with common accounting practices. Additionally, the datasets could be tailored to specific areas of accounting, such as tax accounting, cost accounting e.t.c, to improve the precision in these areas.

Deploying larger models. Deploying larger models could improve the precision in generating scenarios and transactions. Models with more parameters, have more capacity to learn complex patterns. This could be particularly beneficial in accounting tasks, where details and accuracy are crucial. With their increased capacity, larger models may also be better suited to handle the intricacies of financial data, potentially reducing the frequency of incomplete or inaccurate outputs observed with smaller models. However, the larger models come with an increased computational requirement and longer inference times, increasing the related costs.

Acknowledgments

We would like to thank the reviewers for their helpful suggestions and comments. Our work is funded by the German Federal Ministry of Education and Research (BMBF) under the grant agreements 01|S23013C (More-with-Less), 01|S23015A (AI4SCM) and 16SV8857 (KIP-SDM). This work is also funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) Project-ID 528483508 - FIP 12, as well as the European Union under the grant project 101079894 (COMFORT - Improving Urologic Cancer Care with Artificial Intelligence Solutions).

References

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. *arXiv preprint arXiv:2103.06333*.

AI@Meta. 2023. CodeLlama Model Card.

AI@Meta. 2024. Llama 3 Model Card.

Airbus. 2024. Airbus SE Unaudited Condensed Interim IFRS Consolidated Financial Information for the three-month period ended 31 March 2024. Accessed: 20 October 2024.

Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning Distributed Representations of Code. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Bayer. 2024. Quarterly Statement First Quarter of 2024. Accessed: 20 October 2024.

Martin Blais. 2023. beancount: Double-Entry Accounting from Text Files. Accessed: 16 October 2024.

Steven M. Bragg. 2012. *Business Ratios and Formulas*, 3rd edition edition. John Wiley & Sons, Ltd.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*.

Deutsche Telekom. 2024. Interim Group Report Q1 2024. Accessed: 20 October 2024.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Scott A Emett, Marc Eulerich, Egemen Lipinski, Nicolo Prien, and David A Wood. 2023. Leveraging ChatGPT for Enhancing the Internal Audit Process – A Real-World Example from a Large Multinational Company. *Available at SSRN 4514238*.

Marc Eulerich and David A Wood. 2023. A Demonstration of How ChatGPT Can be Used in the Internal Auditing Process. *Available at SSRN 4519583*.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1536–1547. Association for Computational Linguistics.

Markus Freitag and Yaser Al-Onaizan. 2017. Beam search strategies for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation, NMT@ACL 2017, Vancouver, Canada, August 4, 2017*, pages 56–60. Association for Computational Linguistics.

Hanchi Gu, Marco Schreyer, Kevin Moffitt, and Miklos A Vasarhelyi. 2023. Artificial Intelligence Co-Piloted Auditing. *Available at SSRN 4444763*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825*.

Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020. Learning and Evaluating Contextual Embedding of Source Code. In *International Conference on Machine Learning*, pages 5110–5121. PMLR.

Alex Kim, Maximilian Muhn, and Valeri V Nikolaev. 2024. Financial statement analysis with large language models. *Chicago Booth Research Paper Forthcoming, Fama-Miller Working Paper*.

Huaxia Li and Miklos A Vasarhelyi. 2023. Applying Large Language Models in Accounting: A Comparative Analysis of Different Methodologies and Off-the-Shelf Examples. *Available at SSRN 4650476*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason T. Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh,

Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. Starcoder: may the source be with you! *Trans. Mach. Learn. Res.*, 2023.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2024. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Mercedes-Benz Group. 2024. Interim Report Q1 2024. Accessed: 20 October 2024.

Simon Michael. 2023. hledger. Accessed: 16 October 2024.

Van L. Parsons. 2017. *Stratified Sampling*, pages 1–11. John Wiley & Sons, Ltd.

SAP. 2024. Quarterly Statement Q1 2024. Accessed: 20 October 2024.

J Schulman, B Zoph, C Kim, J Hilton, J Menick, J Weng, JFC Uribe, L Fedus, L Metz, M Pokorny, et al. 2022. ChatGPT: Optimizing language models for dialogue.

Yue Wang, Hung Le, Akhilesh Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. 2023. Codet5+: Open code large language models for code understanding and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 1069–1088. Association for Computational Linguistics.

Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 8696–8708. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2024. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

John Wiegley. 2023. Ledger. Accessed: 16 October 2024.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 Technical Report. *arXiv preprint arXiv:2407.10671*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. 2023a. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, et al. 2023b. CodeGeeX: A Pre-Trained Model for Code Generation with Multilingual Evaluations on HumanEval-X. *arXiv preprint arXiv:2303.17568*.

A Prompt Engineering

Chain-of-thought prompting. A method to further enhance prompt engineering is following a chain-of-thought prompting approach (Wei et al., 2024). In chain-of-thought prompting a multi-step problem is split into multiple smaller and simpler steps. This results in the model performing better at performing complex problems (Wei et al., 2024).

Prompt protocol. In total, the chain of thought prompts consists of nine prompts that guide the models in performing their tasks. We list these next.

A.0.1 Prompt 1: Role Definition

This prompt explains the models' role as an auditor. To reduce the inference cost, the model is also instructed to return a short sentence as confirmation that it understands it's task. This also ensures that the model does not generate a different long output. We empirically observed that models have to be specifically asked not to provide a repetition of the provided rules and not to provide a confirmation. Otherwise the models would generate verbose strings, increasing inference time.

A.0.2 Prompt 2: Task Explanation

The second prompt provides the model with an explanation of the tasks it has to perform: a generation of realistic scenarios that affect financial ratios based on a given balance sheet and the generation

of Beancount transactions based on the financial scenarios. The task execution prompt, in which the financial ratio and number of scenarios are given, is also explained in this prompt. Furthermore, we do not provide any explicit formulas for the financial ratios.

A.0.3 Prompt 3: Input Data Explanation

This prompt explains the financial data that is to be processed by the model. It explains the data origin. The statement is a report generated by Beancount, and each line represents a different account to be used in the scenario generation task.

A.0.4 Prompt 4: Output Data Explanation

The fourth prompt outlines the two types of outputs expected from the model. The first output is for the first task, where the model generates scenarios that influence the given financial ratio. The second output is created for the transaction generation task that is based on the first output.

A.0.5 Prompt 5: Plain Text Accounting Rules

This prompt provides the model additional rules that it has to adhere to while solving the tasks. During the prompt engineering process, the models ignored the principles of double-entry bookkeeping. Occasionally models would create transactions that only affect a single account, whereas a transaction has to influence at least two accounts. Liabilities and equity were also problematic, because these accounts increase with negative signs in Beancount. The first rule provides the model with knowledge about double-entry bookkeeping in plain text transactions such as Beancount. The following rule is that the scenarios have to state affected accounts clearly. The third rule states that liabilities and equity increase with negative signs. In the last rule, the model is forbidden to omit transactions during generation and has to generate as many transactions as scenarios.

A.0.6 Prompt 6: Input-Output Example

In this prompt the model is provided with an example on what kind of input it receives and what kind of output is expected. Providing an example of the output, allows for a standardization of the output layouts. The input is a balance sheet generated by Beancount. The output examples are specific to the ratio that the model is tasked to influence.

```
2024-01-01 open Assets:Current:Cash EUR
2024-01-01 open Liabilities:Current:VISA EUR

2024-05-29 * "Withdrawing from ATM with CC"
  Assets:Current:Cash         500 EUR
  Liabilities:Current:VISA   -500 EUR
```

Figure 3: Beancount DSL example for opening an account and withdrawing cash from an ATM with a credit card.

A.0.7 Prompt 7: Balance Sheet Context

A balance sheet, output by Beancount, is provided to the model as context by this prompt. The Beancount balance sheet report is used, so the models have the actual Beancount account names as context. Experiments have shown that otherwise different naming conventions are used, leading to compilation issues for Beancount.

A.0.8 Prompt 8: Scenario Generation

The task outlined in the prompt asks the model to generate 20 scenarios based on the balance sheet provided.

A.0.9 Prompt 9: Transaction Generation

This prompt asks the model to execute the transaction generation task 20 times, which draws on the knowledge of the previous model outputs.

B Beancount DSL

Each transaction in Beancount adheres to a consistent syntax and is entered using a uniform standardised format. A Beancount text file typically comprises numerous transactions, which are then parsed by a compiler. Prior to the execution of transactions, the affected accounts must be open or Beancount returns an unknown account error. The format of the open directive follows the following syntax: `YYYY-MM-DD open Account (optional currency constraint)`

Transactions start with the date of the transaction and are followed by a memo that is provided as an identifier or description. After the memo, two or more postings follow, that specify the affected accounts and the amounts by which they change. An example of a transaction alongside with the opening of the accounts is shown in Figure 3.

The example starts with the opening of two accounts: an assets account for cash and a liabilities account for short term credit card debt (VISA). The liabilities account is debited with 500 euros, which is shown with a negative sign, while the

assets account is credited with the same amount (with the opposite sign). In essence this transaction summarizes the account movements analogous to withdrawing cash from an ATM using a credit card.

C Opening balances in a Beancount file

Figure 4 shows the data included in the Airbus `Beancount` file, which is used to produce the balance sheet report that is fed as context to the LLMs. The generated transactions are appended to this file and subsequently verified by the `Beancount` compiler.

```
; Opening balances
2024-01-01 open Assets:Current:Inventories
2024-01-01 open Assets:Current:TradeReceivables
2024-01-01 open Assets:Current:PortionOfOtherLongTermFinancialAssets
2024-01-01 open Assets:Current:ContractAssets
2024-01-01 open Assets:Current:OtherFinancialAssets
2024-01-01 open Assets:Current:OtherAssets
2024-01-01 open Assets:Current:TaxAssets
2024-01-01 open Assets:Current:Securities
2024-01-01 open Assets:Current:CashAndCashEquivalents

2024-01-01 open Assets:NonCurrent:IntangibleAssets
2024-01-01 open Assets:NonCurrent:PropertyPlantAndEquipment
2024-01-01 open Assets:NonCurrent:InvestmentProperty
2024-01-01 open Assets:NonCurrent:InvestmentsAccountedUnderEquityMethod
2024-01-01 open Assets:NonCurrent:OtherInvestmentsAndOtherLongTermFinancialAssets
2024-01-01 open Assets:NonCurrent:ContractAssets
2024-01-01 open Assets:NonCurrent:OtherFinancialAssets
2024-01-01 open Assets:NonCurrent:OtherAssets
2024-01-01 open Assets:NonCurrent:DeferredTaxAssets
2024-01-01 open Assets:NonCurrent:Securities
2024-01-01 open Assets:HeldForSale

2024-01-01 open Liabilities:Current:Provisions
2024-01-01 open Liabilities:Current:ShortTermFinancingLiabilities
2024-01-01 open Liabilities:Current:TradeLiabilities
2024-01-01 open Liabilities:Current:ContractLiabilities
2024-01-01 open Liabilities:Current:OtherFinancialLiabilities
2024-01-01 open Liabilities:Current:OtherLiabilities
2024-01-01 open Liabilities:Current:TaxLiabilities
2024-01-01 open Liabilities:Current:DeferredIncome

2024-01-01 open Liabilities:NonCurrent:Provisions
2024-01-01 open Liabilities:NonCurrent:LongTermFinancingLiabilities
2024-01-01 open Liabilities:NonCurrent:ContractLiabilities
2024-01-01 open Liabilities:NonCurrent:OtherFinancialLiabilities
2024-01-01 open Liabilities:NonCurrent:OtherLiabilities
2024-01-01 open Liabilities:NonCurrent:DeferredTaxLiabilities
2024-01-01 open Liabilities:NonCurrent:DeferredIncome

2024-01-01 open Liabilities:HeldForSale

2024-01-01 open Equity:CapitalStock
2024-01-01 open Equity:SharePremium
2024-01-01 open Equity:RetainedEarnings
2024-01-01 open Equity:AccumulatedOtherComprehensiveIncome
2024-01-01 open Equity:TreasuryShares
2024-01-01 open Equity:NonControllingInterests

; Opening balances as of 03/31/2024
2024-03-31 * "Opening Balances as of 03/31/2024"
  Assets:Current:Inventories                                        37,656 EUR
  Assets:Current:TradeReceivables                                    4,959 EUR
  Assets:Current:PortionOfOtherLongTermFinancialAssets                 836 EUR
  Assets:Current:ContractAssets                                      1,923 EUR
  Assets:Current:OtherFinancialAssets                                1,831 EUR
  Assets:Current:OtherAssets                                         3,633 EUR
  Assets:Current:TaxAssets                                             618 EUR
  Assets:Current:Securities                                          1,845 EUR
  Assets:Current:CashAndCashEquivalents                             13,615 EUR
  Assets:HeldForSale                                                    52 EUR
  Assets:NonCurrent:IntangibleAssets                                17,055 EUR
  Assets:NonCurrent:PropertyPlantAndEquipment                       17,360 EUR
  Assets:NonCurrent:InvestmentProperty                                  35 EUR
  Assets:NonCurrent:InvestmentsAccountedUnderEquityMethod            2,269 EUR
  Assets:NonCurrent:OtherInvestmentsAndOtherLongTermFinancialAssets 4,955 EUR
  Assets:NonCurrent:ContractAssets                                      62 EUR
  Assets:NonCurrent:OtherFinancialAssets                               721 EUR
  Assets:NonCurrent:OtherAssets                                      1,994 EUR
  Assets:NonCurrent:DeferredTaxAssets                                3,374 EUR
  Assets:NonCurrent:Securities                                       7,964 EUR
  Liabilities:Current:Provisions                                    -4,125 EUR
  Liabilities:Current:ShortTermFinancingLiabilities                 -3,393 EUR
  Liabilities:Current:TradeLiabilities                             -14,202 EUR
  Liabilities:Current:ContractLiabilities                          -27,125 EUR
  Liabilities:Current:OtherFinancialLiabilities                     -2,707 EUR
  Liabilities:Current:OtherLiabilities                              -4,364 EUR
  Liabilities:Current:TaxLiabilities                                  -697 EUR
  Liabilities:Current:DeferredIncome                                  -528 EUR
  Liabilities:NonCurrent:Provisions                                 -5,515 EUR
  Liabilities:NonCurrent:LongTermFinancingLiabilities              -10,286 EUR
  Liabilities:NonCurrent:ContractLiabilities                       -23,540 EUR
  Liabilities:NonCurrent:OtherFinancialLiabilities                  -7,042 EUR
  Liabilities:NonCurrent:OtherLiabilities                             -410 EUR
  Liabilities:NonCurrent:DeferredTaxLiabilities                       -249 EUR
  Liabilities:NonCurrent:DeferredIncome                                -40 EUR
  Liabilities:HeldForSale                                              -74 EUR
  Equity:CapitalStock                                                 -793 EUR
  Equity:SharePremium                                               -4,080 EUR
  Equity:RetainedEarnings                                          -16,674 EUR
  Equity:AccumulatedOtherComprehensiveIncome                        +2,949 EUR
  Equity:TreasuryShares                                               +174 EUR
  Equity:NonControllingInterests                                       -36 EUR
```

Figure 4: Example Beancount file.