SEC-QA: A Systematic Evaluation Corpus for Financial QA

Viet Dac Lai, Michael Krumdick, Charles Lovering
Varshini Reddy, Craig Schmidt, Chris Tanner
Kensho Technologies
{chris.tanner}@kensho.com

Abstract

The financial domain frequently deals with large numbers of long documents that are essential for daily operations. Significant effort is put towards automating financial data analysis. However, a persistent challenge, not limited to the finance domain, is the scarcity of datasets that accurately reflect real-world tasks for model evaluation. Existing datasets are often constrained by size, context, or relevance to practical applications. Moreover, LLMs are currently trained on trillions of tokens of text, limiting access to novel data or documents that models have not encountered during training for unbiased evaluation. We propose SEC-QA, a continuous dataset generation framework with two key features: 1) the semi-automatic generation of Question-Answer (QA) pairs spanning multiple long context financial documents, which better represent real-world financial scenarios; 2) the ability to continually refresh the dataset using the most recent public document collections, not yet ingested by LLMs. Our experiments show that current retrieval augmented generation methods systematically fail to answer these challenging multi-document questions. In response, we introduce a QA system based on program-of-thought that improves the ability to perform complex information retrieval and quantitative reasoning pipelines, thereby increasing QA accuracy.

1 Introduction

Large Language Models (LLMs) have demonstrated impressive capabilities in a wide range of natural language processing (NLP) applications (Brown et al., 2020). Even though LLMs were scaled to an unprecedentedly large number of parameters, they still face many issues with hallucinations (Ji et al., 2023), poor reading comprehension (Liu et al., 2024), and private data leakage (Balloccu et al., 2024). To address these issues, Retrieval Augmented Generation (RAG) produces responses by using a few retrieved documents from reliable sources. As a result, it offers more dependable answers with fewer hallucinations, even when employing much smaller language models (Borgeaud et al., 2022). Evaluating RAG-based systems is challenging due to the added complexity of the retrieval step (Chen et al., 2024), and the cascading effect of the upstream retrieval task on the downstream QA task. Early benchmarks for RAG-based systems mainly focus on simple common-sense questions that can be answered by retrieving a single piece of text from a single knowledge source (Joshi et al., 2017; Dunn et al., 2017). Often these knowledge sources were open textual knowledge bases such as Wikipedia (Yang et al., 2018) and WikiHow (Deng et al., 2020).

These benchmarks face two data leakage issues. First, they were created from open Internet sources (Xue et al., 2021; Penedo et al., 2023), e.g., Wikipedia, which is heavily used in LLM pre-training (Touvron et al., 2023). Second, the static benchmarks themselves are leaked to the Internet (Balloccu et al., 2024). LLM training datasets subsequently include these benchmarks, leading to inflation and unreliable results on these benchmarks. To combat these problems, there have been proposals to keep benchmarks private (Mialon et al., 2023) or updated regularly (Fan et al., 2023).

More importantly, in many financial applications, evaluation based on common-sense knowledge and open knowledge sources may not reflect the system's true capabilities (Zhang et al., 2024). The questions in professional use are far more complex, requiring some combination of multi-hop reasoning (Yang et al., 2018), multi-source reference (Tang and Yang, 2024), document-structure reference (Saad-Falcon et al., 2023), and collection structure reference. Additionally, knowledge bases in a specific domain may be less diverse than open knowledge bases, e.g., Wikipedia, leading to poor retrieval performance which in turn has a severe

impact on the LLM's response. Retrieval in some domain-specific areas may instead require domain knowledge to achieve effective retrieval that has not been captured in open-domain evaluation.

To overcome these issues, this work introduces a framework for designing practical quantitative questions. These questions are much more challenging than existing financial QA tasks such as FinQA(Chen et al., 2021) or TAT-QA (Zhu et al., 2021). The framework allows us to customize questions at the needed complexity for the target applications with potential variety in question complexity including *multiple entities/financial periods*, *multi-hop reasoning*, *document structure*, *collection structure*, and *multiple outputs*. We leverage Internet-accessible document collections, and open tabular databases to create real-world complex quantitative questions in finance. We evaluate four RAG-based systems and show that RAG systems systematically fail on these carefully designed real-world questions. Moreover, we show that recent LLMs can use code to effectively navigate the structure of the document collections; e.g., that earnings per share information is within 10-Ks, and that there are 10-Ks for each company and fiscal year. This leads to drastically improved levels of performance. Additionally, this framework can be used to dynamically refresh the benchmarks regularly to prevent training data leakage.

The contributions of this paper are:

- A framework (SEC-QA) for dynamically generating quantitative multi-hop QA tasks for the financial domain from publicly accessible documents and databases.
- A set of practical and challenging questions for Quantitative Reasoning in the financial domain that vanilla RAG models systematically fail to answer.
- A system that utilizes program-of-thought and the rich structure of the document collection to improve QA performance.

2 Related Work

Reasoning Capabilities of LLM has been found in natural language processing and other fields. Some LLMs exhibit emergent capabilities if they are large enough. A simple prompt "*Let's think step by step*" causes a model to generate solutions with reasoning steps in a chain-of-thought (Wei et al., 2022). More advanced prompting

techniques have been discovered which are similar to human reasoning processes such as tree-of-thoughts (Yao et al., 2024) and self-verification (Weng et al., 2023). However, the numerical reasoning of LLMs is still limited, motivating the adoption of programming languages to offload numerical tasks in program-of-thought (Chen et al., 2023) and program-synthesis (Austin et al., 2021). Reasoning capability is enhanced in additional training on reasoning through human feedback (Ouyang et al., 2022). However, LLMs still struggle with many domain-specific tasks such as finance (Koncel-Kedziorski et al., 2023). Recent studies have pointed out that many popular benchmarks are contained in LLM pre-training data (Riddell et al., 2024), which causes the inflation of model performance. As such, benchmarks are kept private (Mialon et al., 2023), or updated regularly (Fan et al., 2023) to mitigate data contamination.

Document Grounded Quantitative Reasoning involves numerical extraction and numerical reasoning. Previous work in NLP has explored numerical extraction from scientific documents (Harper et al., 2021; Elazar et al., 2019) and financial documents (Loukas et al., 2022). Existing datasets for the financial domain that require quantity extraction include HybridQA (Chen et al., 2020), TAT-QA (Zhu et al., 2021), MultiHierTT (Zhao et al., 2022), FinQA (Chen et al., 2021), and ConvFinQA (Chen et al., 2022). However, these works only involve a small amount of grounding context (e.g., a single page, a single document).

Multi-Document QA: TriviaQA (Joshi et al., 2017) and SearchQA (Dunn et al., 2017) require the model to search over a large collection of documents. However, the question itself can be answered by reading a few sentences extracted from a single document. Some multi-document QA datasets were created for open-ended QA such as summarization (MultiNews (Fabbri et al., 2019), WikiHowQA (Bolotova-Baranova et al., 2023)) where skimming over given documents and extracting evidential cues from these documents are essential. The HotpotQA (Yang et al., 2018) dataset specifically targets multi-hop questions to resolve hidden cross-document reference entities in the questions. However, these datasets were collected from open knowledge bases (e.g., Wikipedia), so they have most likely been leaked in LLM pre-training data (Touvron et al., 2023).

Multihop-RAG (Tang and Yang, 2024) propose

	Dataset	Multi-Doc	Multi-Hop	Refreshable	#Test	Context	#Docs	Data
General	HybridQA	✓	✓	-	3,463	2,326	44	Hybrid
	TriviaQA	✓	-	-	17,210	3,760	486,956	Text
	SearchQA	-	-	-	27,248	38	-	Text
	HotPotQA	✓	✓	-	7,405	928	15,519	Text
Finance	TAT-QA	-	-	-	1,669	47	-	Hybrid
	MultiHiertt	-	-	-	1,566	1,646	-	Hybrid
	FinQA	-	-	-	1,147	628	-	Hybrid
	ConvFinQA	-	-	-	434	628	-	Hybrid
	Multihop-RAG	✓	-	-	2,556	1,574	609	Text
	SEC-QA	✓	✓	✓	Flexible	123,000	1,315	Hybrid

Table 1: Comparison of QA benchmarks in the quantitative and finance domain. Refreshable indicates that the dataset can be automatically renewed/generated with a different document set. Hybrid indicates that the context contains both tabular and textual data.

a multi-hop dataset for financial documents that differs from our work in the following ways: (i) their work studies multi-hop reasoning only with regards to parallel retrieval queries, while we consider both parallel and sequential reasoning steps; (ii) the news documents used to create their dataset do not reflect the real-world sources such as official filings used by financial professionals, that often span hundreds of pages.

Financial NLP has explored non-quantitative tasks such as named entity recognition (Salinas Alvarado et al., 2015), sentiment analysis (Malo et al., 2014), classification (Sinha and Khandait, 2021), question answering (Maia et al., 2018), boundary detection (Au et al., 2021), and entity/event extraction Lu et al. (2023).

3 Framework Construction

We propose Systematic Evaluation Corpus for Financial QA (SEC-QA), a framework for generating financial Multi Document Questions and Answers (MDQA). We also refer to the questions generated by this framework with the same name, SEC-QA.

3.1 Task Definition

MDQA is defined as follows: A system S is asked a question q with answer a. q can be answered by looking in at the document collection $C = \{D_i | 1 \leq i \leq N\}$. Each document consists of several pages $p_i = (t_{ij}, c_{ij})$ where t_{ij}, c_{ij} are the title and the content, respectively.

3.2 Resources

SEC-QA, a framework for flexibly creating questions for MQDA, requires: 1) Database T with values partitioned by variables of interest (e.g., revenue partitioned by company and fiscal year). 2) Document collection C that contains the information needed to compute the values within T.

Specifically, we leverage private-sector financial data from market-trusted sources, ensuring comprehensive and accurate datasets. We collect key financial metrics and their associated documents to create a database (key-value-document table) $T \in (c, y, k, v, d)$, where v represents the value of the key metric k for company c in fiscal year y as mentioned in document d. Because T tracks the documents in C that source each value then in addition to question accuracy it is possible to report document and page-level retrieval metrics.

Our collection comprises 10 metrics for 18 publicly traded companies from the S&P 500 list from 2010 to 2023. We also collect their annual reports (Form 10-K), quarterly reports (Form 10-Q), and unscheduled event reports (Form 8-K) for the same period. Since the documents are published in HTML format, we convert the documents into PDF(s), and then we parse the PDF(s) into JSON format using a public PDF extraction service [1]. Documents are represented as JSON objects with paragraphs, well-structured tables, and machine-detected titles.

---

[1] https://kensho.com/extract

3.3 Question Complexity

Question design is an important step for a successful QA system evaluation. In this work, we identify several factors that increase the complexity of a question in the financial domain.

We define an **atomic question** as a question that seeks a single piece of information that can be extracted directly from a single document. Such questions usually involve a single entity for a particular financial period, e.g., *"What is the total revenue of Apple Inc. for the fiscal year 2022?"*.

However, in financial analysis, questions tend to be much more complex. They require extracting multiple pieces of data, transforming the extracted data, and presenting the answer in various formats. Further, we have observed the following main challenges in multiple-document QA for finance:

- **Parallel Reference** questions require the same kind of information over multiple entities or time periods, e.g., *"What is Intel's revenue growth in the last 5-year period?"*. This requires extracting information from a few documents. The complexity of parallel reference can be measured by the number of entities/years needed to answer the question.
- **Multi-hop Reference** questions require reference resolution of a few implicitly defined entities through some deterministic constraints e.g. *"Show the 5-year stock price history of the top 5 most valuable companies in the S&P 500 index"*. The complexity of multi-hop questions is usually measured by the number of hops needed to answer the question and the complexity of the constraints.
- **Structural Reference** involves document structure reference and collection structure reference. Document structure reference refers to a particular section/table/figure in a document (Saad-Falcon et al., 2023). Collection structure reference refers to a subset of documents in the collection (e.g., *"recently quarterly filings", "their earning calls"*), narrowing the document search space.
- **Multi-Output** questions expect multiple values (e.g., *"Analyze the financial performance of the top 3 competitors of Amazon.com, Inc. in terms of revenue for the last 4 quarters, by computing their revenue growth, gross margin, and operating margin."*). To our knowledge, this type of question has not been addressed in previous work.

3.4 Question Design

Question Template: Previous work in QA has tackled some challenges in QA such as multi-hop (Yang et al., 2018), and document structure (Saad-Falcon et al., 2023) to target some specific complex question types. Our framework allows many question types including parallel reference, multi-hop reference, document structure, collection structure reference, and multi-output questions. Moreover, due to the rich information in the database, we can design questions to require finance domain jargon (e.g., using a company's stock symbol to refer to the company) and language regularities (e.g., omitting financial periods if asking for the latest figures). Using this framework, we can flexibly opt for different combinations of complexity (e.g., parallels and multi-hop in the same question).

Template filling: The question template can be filled semi-automatically through a simple rule-based system that randomly picks the entities, metrics, and periods from the database. This allows control over the complexity, quality, and distribution of the generated questions and answers.

4 Experiment

This section presents our findings across three use cases of the SEC-QA framework.

4.1 QA Systems

We evaluate 4 systems with different characteristics for a broad understanding of MDQA in finance:

- **Vanilla RAG**: To demonstrate the challenge of answering complex questions, we employed a simple retrieval-based system with direct generation.
- **Multi Query RAG**: leverages multiple queries for a given input question.
- **CodeGen+PageR**: We employ an LLM to generate code that makes use of two helper functions: *"retrieve_relevant_pages"* retrieves k pages from the whole collection; *"extract_value"* calls a prompted LLM to extract the value from the given retrieved pages. This system allows an LLM to decompose a complex question into atomic questions as well as make a full plan of how to answer this question. In turn, it allows us to examine the planning capability of the LLM.
- **CodeGen+DocS+PageR**: Financial document collections track meta information for

Function	Description
select_document(stock_symbols, form_types, fiscal_years)	Select a few documents given some filters
retrieve_relevant_pages(text_query, documents)	Retrieve top-k pages from the given documents
extract_value(text_query, pages)	Extract a value and its corresponding multiplier from the given pages

Table 2: List of helper functions that the CodeGen systems can use

each document like company, fiscal year, and form type. This can be used to reduce the retrieval search space. Beyond the two functions *"retrieve_relevant_pages"* and *"extract_value"*, we introduce a *"select_document"* function that filters the document collection based on meta information such as the company stock symbol and fiscal year. Critically, a document can be selected and then queried for pages.

We use OpenAI's Ada as the neural embedding for retrieval and GPT4 (gpt-4-1106-preview) as the LLM for all our experiments. We use the same three questions as the exemplars for the CodeGen system, varying only the available helper functions. We test different numbers of retrieved pages ($k \in [4, 128]$) and report the best performance.

4.2 Evaluation

Due to the complexity of the pipeline and models we used, we evaluate the model's performance in 3 stages: document retrieval, page retrieval, and question answering. For document retrieval and page retrieval, we report the Precision@K, Recall@K, and F1@K. For systems that only involve page retrieval, we report their document retrieval performance based on the document signature of the retrieved pages. Since many numbers in financial reports are rounded to various levels (thousands, millions, billions), using exact matches for automatic answer scoring is challenging. Therefore, we accept an answer if its value is within a 1% margin of error from the golden answer.

4.3 Single Value Extraction Task

We begin with a simple extraction MDQA task that requires a model to retrieve an exact numeric span (e.g., 1234.5) and unit value (e.g., millions) from a document. For this use case, we use SEC-QA to generate questions based on the following templates: (1) is usually used for the latest update of a metric; (2) is used for the previous financial periods (e.g., years and quarters).
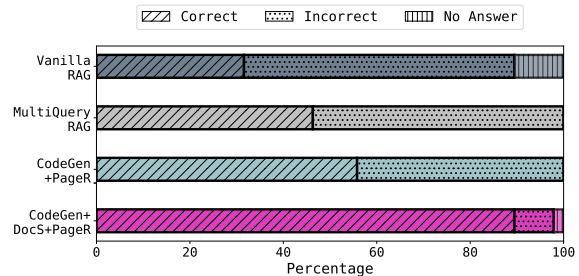

Figure 1: Performance of the simple value extraction

(1)   *What is {**company**}'s {**metric**}?*

(2)   *What is {**company**}'s {**metric**} in {**year**}?*

Financial analysts use this language regularly, in which the year is omitted from the question. As such, a model must catch that regularity to identify the correct value term to extract. To confirm the existence of the metric in the document, we remove the questions whose answers can not be found in the document with simple string matching.

Figure 1 shows the performances of the models of this single-document value extraction task. CodeGen+DocS+PageR system performs best, with 89.5% accuracy. On the other hand, the CodeGen+PageR and Vanilla-RAG systems lag behind with 31.6% and 55.8% accuracy. Comparing the two CodeGen models, we can see that the CodeGen w/ DocS outperforms the variant w/o DocS. This suggests that document selection is important for financial questions. This result also shows that a general neural-based document retrieval struggles to cope with the demanding requirements for retrieval in finance. Part of the reason the neural-based retrieval struggles is because of the structure of the public filings in the financial domain. Many documents are very similar to each other, especially documents for the same company. Long identical phrases are often used for multiple years. Without document selection, LLMs end up having to process irrelevant pages collected from previous years. Section 5.2 shows an example of models without document selection. This experiment shows there is a large performance difference for different pipeline settings.

System	Correct (%)
Vanilla RAG	8.3
Multi Query RAG	16.7
CodeGen+PageR	37.5
CodeGen+DocS+PageR	33.3

Table 3: Performance on compound value extraction.

4.4 Compound Value Extraction Task

Compound or high-order metrics appear frequently in financial analysis. They are usually computed based on a few other metrics reported in the public filings. As such, being able to answer questions with compound metrics is crucial to automating the pipeline of financial analysis. Previous work considers these questions as *NULL* (Tang and Yang, 2024), skipping the question. In real-world applications, a model should provide its best estimation based on the provided information, ideally with an explanation to justify the estimation.

To do this, we design a set of questions with compound metrics, such as Revenue Per Employee (RPE) using the same Templates 1, 2. Some companies report these metrics in their filings, so we only consider metrics that cannot be easily found with string matching. Thus, the model must be able to understand the formula to calculate to answer correctly. In total, we generated 24 of these questions. Because the value is not explicitly in the text, these questions require both *a priori* knowledge of the metrics and additional reasoning steps to compute the metric. Therefore, we expect these questions to be more difficult.

Table 3 shows the performance of the model on this question set. Performance for all tested systems on these Compound Value Extraction questions is significantly lower than the Single Value Extraction questions (as presented in Section 4.3).

For this task, we are able to investigate the performance at the sub-metric level thanks to the sub-metric data in the database. The Vanilla RAG and Multi Query RAG models do not extract value at the sub-metric level, so we omit these models from this analysis. We find that CodeGen-based models systematically query each sub-metric (e.g., Long-Term Debt, and Long-Term Leases). However, once the model unrolls the main metric (e.g., total debt) into sub-metrics, many sub-metrics are missing from the document due to two reasons: (1) the sub-metrics are also compound metrics and (2) some sub-metrics do not apply to some companies. These lead to a false extraction or duplication (Long-Term Debt being extracted twice for Long-Term Debt and Long-Term Leases).

This test highlights the difficulty of the value extraction task in the financial domain. This also shows how we can easily use our framework to customize the test without data annotation.

4.5 Multi-Document QA Task

To measure the performance on more complex question, we design a set of question templates that refer to metrics of different years and companies as follows:

(3) *How much common dividends did **{company}** pay in the last **{num_year}** years in US dollars?*

(4) *What is the percentage difference of **{company1}**'s **{metric}** compared to that of **{company2}**?*

(5) *What is **{company}**'s overall revenue growth over the last **{num_year}**-year period?*

(6) *Among **{company_names}**, what is the **{metric2}** of the company that has the highest **{metric1}**?*

Figure 2 shows the accuracy for the three models on this task with different numbers of retrieved pages k. We can see that CodeGen+DocS+PageR outperforms the other models with a high margin, correctly answering 108 of 135 questions ($80\%, k = 32$) compared to 71 with CodeGen+PageR ($52\%, k = 48$) and only 41 with Vanilla RAG ($30\%, k = 32$). Notably, CodeGen+DocS+PageR outperforms the other two models with a very low number of retrieved pages ($k = 4$) with 74 correct answers (55%).

More importantly, CodeGen models are more responsive to the number of retrieved pages. CodeGen+DocS+PageR performance improves rapidly when the number of retrieved pages k increases from 4 to 32, whereas CodeGen+PageR improves at a lower rate, and Vanilla RAG barely improves. This suggests that the Vanilla RAG pipeline is bottlenecked at the retrieval step, which we analyze in depth in Section 5.1.
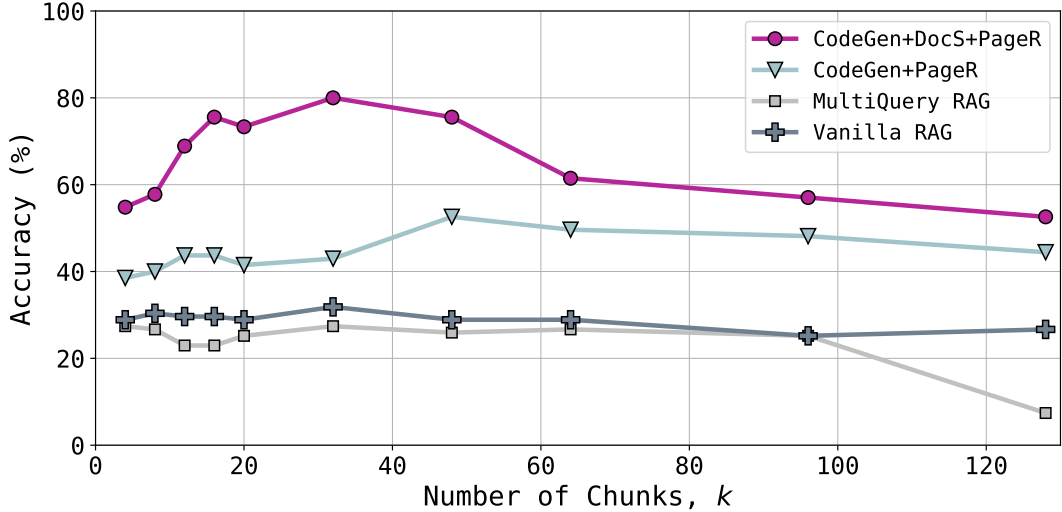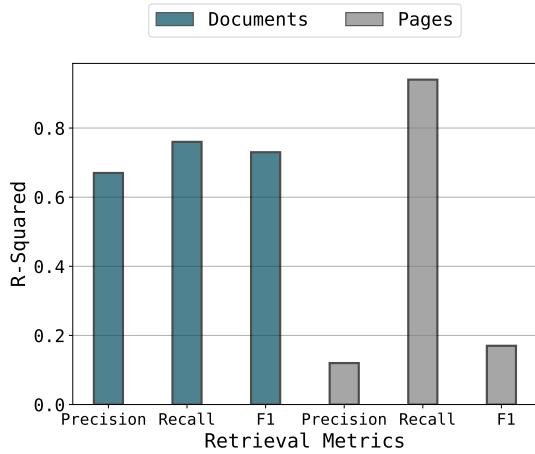
Figure 2: Performance of systems on Multi-Doc QA.

Figure 3: R^2 between accuracy and retrieval metrics.

5 Discussion

5.1 System Bottlenecks

Previous sections highlight how additional retrieval capabilities improve LLM performance on MDQA. In this section, we perform further analysis to identify the main performance bottleneck across the different systems. We reuse the Template 6 which has both parallel and multi-hop references. Beyond document/page retrieval and task performance, we also compute the coefficient of determination R^2 between these performances.

Figure 3 shows the R^2 values between the system accuracy and retrieval metrics at both document and page levels. Recall has the highest R^2 values, with 0.94 and 0.76 for page and document levels respectively, suggesting that recall performance

is crucial to the accuracy of the multi-document QA performance. On the other hand, page-level precision has a low R^2 value of 0.12, indicating a weak correlation with overall performance.

From Figure 4, we observe that the QA accuracy of a system is directly proportional to document and page-level recall scores, supporting our previous claim about the correlation between recall and accuracy. Specifically, we note that a model's accuracy aligns more closely with the page-level recall score. The Vanilla RAG model lags behind the CodeGen-based models in terms of accuracy. However, when it is allowed to use multiple retrieval queries (Multi Query RAG), we observe an improvement in retrieval performance, which consequently leads to an increase in QA accuracy.

CodeGen+PageR has a higher recall and QA performance compared to Multi Query and Vanilla RAG. This is due ability of the CodeGen models to break down complex questions into atomic ones and systematically retrieve pages based on the atomic questions. CodeGen+DocS+PageR is observed to be the best model. We attribute this to the addition of the rule-based document selection step, which effectively retrieves relevant documents, thus improving document-level recall.

Overall this experiment shows that our dataset and framework provide us a useful tool to examine multi-document QA in detail. This gives us a better signal for future improvement of the pipeline.
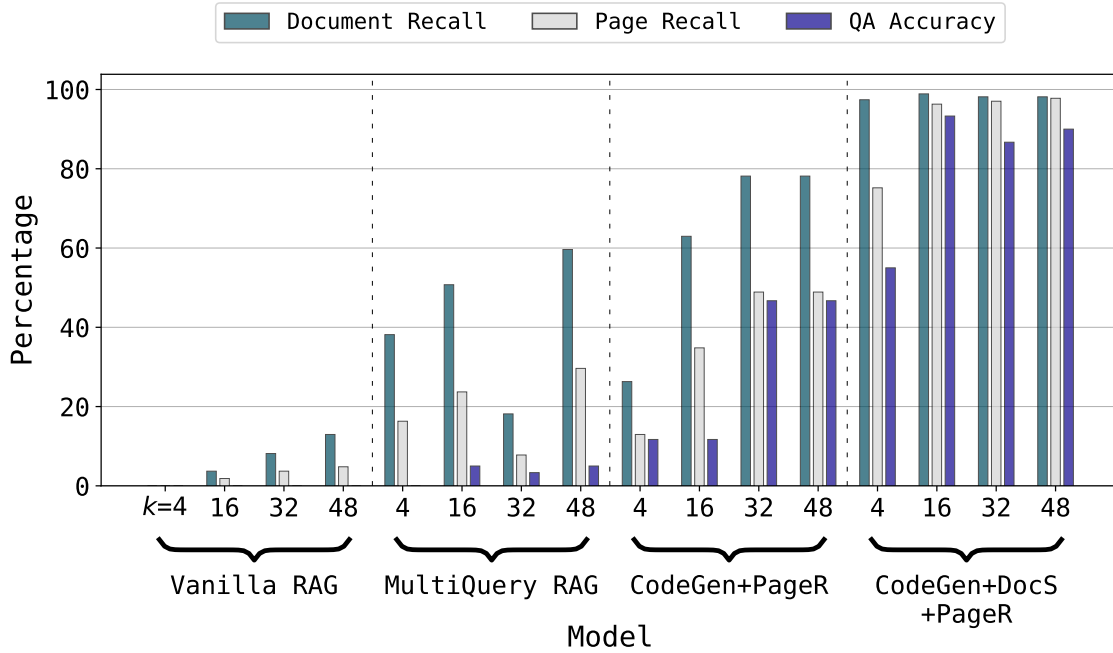
Figure 4: Document-level recall, page-level recall, and QA accuracy of all examined systems with varying k values.

5.2 Case study

Table 4 shows the top 4 retrieved pages across QA systems for *"What is Adobe's total number of employees reported in 2022?"*. Only Code-Gen+DocS+PageR successfully retrieves the page containing the golden answer. Although Vanilla RAG retrieves Adobe's 10-K pages 3 of 4 times, the fiscal year is consistently wrong. Multi Query RAG and CodeGen+PageR also retrieve the wrong fiscal year. This shows how financial documents can easily confuse modern retrieval systems.

5.3 Stability Tests

One of the main objectives of this work is to provide a robust benchmark that prevents performance inflation through data leaks by dynamically generating QA pairs using the latest financial documents. This raises the question of whether evaluation scores on a new benchmark version are comparable to those on former versions. To measure the benchmarking consistency, we construct five distinct versions of the dataset using data from different years (2019 to 2023) while keeping the question templates and the set of companies constant to maintain a consistent difficulty level.

The accuracy of the models on different year-based sets varied marginally with small standard deviation ($1.3\% < \sigma < 2.0\%$). This shows that we can reliably compare the performance of models

between different versions of the benchmark. We show the detailed result in Figure 5 (Appendix ??).
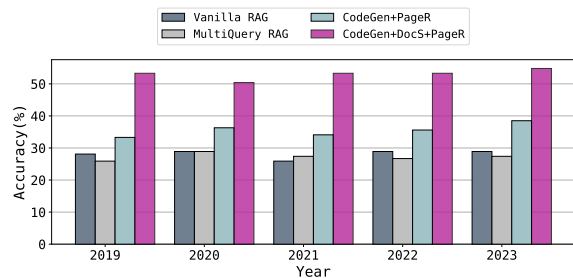

Figure 5: Performance of models on different variants of the same dataset.

5.4 Execution Cost

While code generation-based systems offer superior performance, these models have a higher operational cost and latency. We compute the average number of LLM calls across the four examined systems. From Figure 6, CodeGen systems require approximately four times more calls than Vanilla and Multi Query RAG systems, which can potentially quadruple operational costs and latency. One proposed solution to mitigate latency is to parallelize LLM calls. However, the iterative nature of multi-hop questions poses a challenge to effective parallelization strategies.

Model	Matched	Question: *"What is Adobe's Total Employees reported in 2022?"*
Vanilla RAG	N	Page: 15 Form type: 10-K; Company: adobe; Fiscal year: ~~2015~~; Period end date: 2015-11-27
	N	Page: 15 Form type: 10-K; Company: adobe; Fiscal year: ~~2014~~; Period end date: 2014-11-28
	N	Page: 29 Form type: ~~8-K~~; Company: adobe; Fiscal year: ~~2020~~; Period end date: 2020-12-07
	N	Page: 35 Form type: 10-K; Company: adobe; Fiscal year: ~~2012~~; Period end date: 2012-11-30
Multi Query RAG	N	Page: 15 Form type: 10-K; Company: adobe; Fiscal year: ~~2015~~; Period end date: 2015-11-27
	N	Page: 15 Form type: 10-K; Company: adobe; Fiscal year: ~~2014~~; Period end date: 2014-11-28
	N	Page: 35 Form type: 10-K; Company: adobe; Fiscal year: ~~2012~~; Period end date: 2012-11-30
	N	Page: 18 Form type: 10-K; Company: adobe; Fiscal year: ~~2020~~; Period end date: 2020-11-27
CodeGen + PageR	N	Page: 15; Form type: 10-K; Company: ADBE; Fiscal year: ~~2015~~; Period end date: 2015-11-27
	N	Page: 18; Form type: 10-K; Company: ADBE; Fiscal year: ~~2023~~; Period end date: 2023-12-01
	N	Page: 16; Form type: 10-K; Company: ADBE; Fiscal year: ~~2021~~; Period end date: 2021-12-03
	N	Page: 15; Form type: 10-K; Company: ADBE; Fiscal year: ~~2014~~; Period end date: 2014-11-28
CodeGen + DocS + PageR	N	Page: 16; Form type: 10-K; Company: ADBE; Fiscal year: 2022; Period end date: 2022-12-02
	N	Page: 2; Form type: 10-K; Company: ADBE; Fiscal year: 2022; Period end date: 2022-12-02
	Yes	Page: 15; Form type: 10-K; Company: ADBE; Fiscal year: 2022; Period end date: 2022-12-02
	N	Page: 38; Form type: 10-K; Company: ADBE; Fiscal year: 2022; Period end date: 2022-12-02

Table 4: Case study on the inclusion of the page containing the golden answer within the top 4 retrieved pages for the question, *"What is Adobe's total number of employees reported in 2022?"*

5.5 Automatic Question Design

Thanks to the advancement in code generation, in principle an LLM with access to a database can generate both questions and code necessary to answer the questions. To do that, one can prepare available resources such as documents and functions to access the database and static variable names for relevant entities (e.g. company name, metric name); next, prompt the LLM to generate questions and code that solve the question using the provided functions. Once the LLM generates questions and answers, the generated code is executed to obtain the answer. After that, one can hire a financial expert to verify the question, the code, and the answer. Notably, the challenge remains at test time as that model will *not* have access to the database T, only the document collection C.

The benefit of using this method is the potentially higher diversity in question sets. It also provides the code that is used to solve the question. However, this method still requires manual inspection of the question, the code, and the alignment between the question and the code, which are not trivial. In this work, we did not use this method to generate any of the questions used above.

6 Conclusion

We introduce SEC-QA which we leverage to create questions that current RAG approaches consistently fail to answer. This framework can be used to dynamically generate complex practical
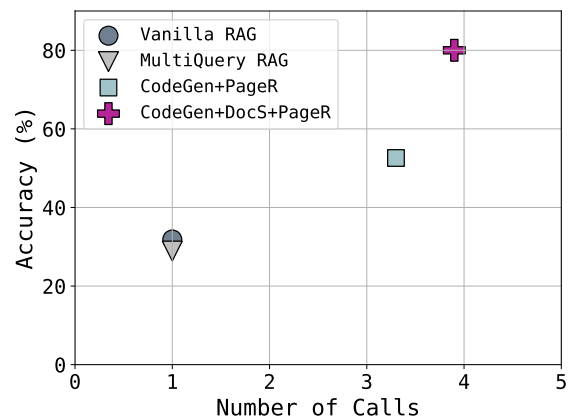
Figure 6: Average number of LLM calls used by the systems in comparison with accuracy.

questions grounded in the financial domain. Our study highlights the challenges posed by retrieval models in handling multi-doc long-context questions and explores strategies to address these bottlenecks. Furthermore, we propose a method based on program-of-thought and RAG designed to enhance retrieval and downstream performance compared to conventional RAG systems.

Limitation

This paper assumes the existence of a collectible set of documents, a tabular dataset of financial metrics, and a method to map these financial metrics to the documents. We currently explore databases in the private sector, where public reports are heavily regulated, making it relatively straightforward to align the documents with the dataset.

However, in the public sector, reports often vary significantly due to inconsistencies in reporting standards. As a result, finding a collection of documents, a corresponding dataset and their alignment is more challenging. For instance, our attempts with the US state government's Annual Comprehensive Financial Report (ACFR) and the US Annual Survey of State Government Tax Collections published by the US Census have proven extremely difficult to reverse-engineer into a usable dataset.

Ethical Consideration

This dataset was generated automatically from an existing financial database without any involvement of human annotators. Although the CodeGen systems demonstrate significant performance improvements, we do not recommend using them as a replacement for traditional financial analysis tools and financial advice.

References

Willy Au, Abderrahim Ait-Azzi, and Juyeon Kang. 2021. Finsbd-2021: the 3rd shared task on structure boundary detection in unstructured text in the financial domain. In *Companion Proceedings of the Web Conference 2021*, pages 276–279.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Simone Balloccu, Patrícia Schmidtová, Mateusz Lango, and Ondrej Dusek. 2024. Leak, cheat, repeat: Data contamination and evaluation malpractices in closed-source LLMs. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 67–93, St. Julian's, Malta. Association for Computational Linguistics.

Valeriia Bolotova-Baranova, Vladislav Blinov, Sofya Filippova, Falk Scholer, and Mark Sanderson. 2023. WikiHowQA: A comprehensive benchmark for multi-document non-factoid question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5291–5314, Toronto, Canada. Association for Computational Linguistics.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

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.

Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17754–17762.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036, Online. Association for Computational Linguistics.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. 2022. ConvFinQA: Exploring the chain of numerical reasoning in conversational finance question answering. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6279–6292, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yang Deng, Wai Lam, Yuexiang Xie, Daoyuan Chen, Yaliang Li, Min Yang, and Ying Shen. 2020. Joint learning of answer selection and answer summary generation in community question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7651–7658.

Matthew Dunn, Levent Sagun, Mike Higgins, V Ugur Guney, Volkan Cirik, and Kyunghyun Cho. 2017. Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*.

Yanai Elazar, Abhijit Mahabal, Deepak Ramachandran, Tania Bedrax-Weiss, and Dan Roth. 2019. How large are lions? inducing distributions over quantitative attributes. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3973–3983, Florence, Italy. Association for Computational Linguistics.

Alexander Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. 2019. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1074–1084, Florence, Italy. Association for Computational Linguistics.

Lizhou Fan, Wenyue Hua, Lingyao Li, Haoyang Ling, Yongfeng Zhang, and Libby Hemphill. 2023. Nphardeval: Dynamic benchmark on reasoning ability of large language models via complexity classes. *arXiv preprint arXiv:2312.14890*.

Corey Harper, Jessica Cox, Curt Kohler, Antony Scerri, Ron Daniel Jr., and Paul Groth. 2021. SemEval-2021 task 8: MeasEval – extracting counts and measurements and their related contexts. In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 306–316, Online. Association for Computational Linguistics.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Rik Koncel-Kedziorski, Michael Krumdick, Viet Lai, Varshini Reddy, Charles Lovering, and Chris Tanner. 2023. Bizbench: A quantitative reasoning benchmark for business and finance. *arXiv preprint arXiv:2311.06602*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12.

Lefteris Loukas, Manos Fergadiotis, Ilias Chalkidis, Eirini Spyropoulou, Prodromos Malakasiotis, Ion Androutsopoulos, and Georgios Paliouras. 2022.

FiNER: Financial numeric entity recognition for XBRL tagging. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4419–4431, Dublin, Ireland. Association for Computational Linguistics.

Dakuan Lu, Jiaqing Liang, Yipei Xu, Qi He, Yipeng Geng, Mengkun Han, Ying Xin, Hengkui Wu, and Yanghua Xiao. 2023. Bbt-fin: Comprehensive construction of chinese financial domain pre-trained language model, corpus and benchmark. *ArXiv*, abs/2302.09432.

Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. 2018. Www'18 open challenge: financial opinion mining and question answering. In *Companion proceedings of the the web conference 2018*, pages 1941–1942.

Pekka Malo, Ankur Sinha, Pekka Korhonen, Jyrki Wallenius, and Pyry Takala. 2014. Good debt or bad debt: Detecting semantic orientations in economic texts. *Journal of the Association for Information Science and Technology*, 65(4):782–796.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for ccon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*.

Martin Riddell, Ansong Ni, and Arman Cohan. 2024. Quantifying contamination in evaluating code generation capabilities of language models. *arXiv preprint arXiv:2403.04811*.

Jon Saad-Falcon, Joe Barrow, Alexa Siu, Ani Nenkova, Ryan A Rossi, and Franck Dernoncourt. 2023. Pdftriage: Question answering over long, structured documents. *arXiv preprint arXiv:2309.08872*.

Julio Cesar Salinas Alvarado, Karin Verspoor, and Timothy Baldwin. 2015. Domain adaption of named entity recognition to support credit risk assessment. In *Proceedings of the Australasian Language Technology Association Workshop 2015*, pages 84–90, Parramatta, Australia.

Ankur Sinha and Tanmay Khandait. 2021. Impact of news on the commodity market: Dataset and results. In *Advances in Information and Communication: Proceedings of the 2021 Future of Information and Communication Conference (FICC), Volume 2*, pages 589–601. Springer.

Yixuan Tang and Yi Yang. 2024. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. *arXiv preprint arXiv:2401.15391*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2023. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2550–2575, Singapore. Association for Computational Linguistics.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Tianjun Zhang, Shishir G Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, and Joseph E Gonzalez. 2024. Raft: Adapting language model to domain specific rag. *arXiv preprint arXiv:2403.10131*.

Yilun Zhao, Yunxiang Li, Chenying Li, and Rui Zhang. 2022. MultiHiertt: Numerical reasoning over multi hierarchical tabular and textual data. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6588–6600, Dublin, Ireland. Association for Computational Linguistics.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, Online. Association for Computational Linguistics.

A   Model Details

Throughout the whole study, we use OpenAI's Ada Embedding to encode texts for retrieval. We use the `Langchain` implementation of vector-based retrievers (i.e., vanilla and multi-query retrievers). We use GPT-4 (gpt-4-0125-preview) as the LLM for both the question answering in RAG-based systems and the code generation and value extraction in CodeGen-based systems.

B   Example of code generated by the CodeGen

Figure 7 shows an example of code generated by the CodeGen+DocS+PageR system. Figure 8 and 9 shows the prompt and few-shot demonstrations used to generate code for the Code-Gen+DocS+PageR systems.

C   Example of question with metric description

What is the Boeing's Total Debt in 2023? Where Total Debt is defined as:

Total Debt is a supplemental line item across all templates with the following components:

Standard template:

Short-term Borrowings

Current Portion of Long-Term Debt

Current Portion of Leases

Long-Term Debt

Long Term Leases

Finance Div. Debt Current

Finance Div. Debt Non-Curr.

Banks template:

Short-Term Borrowings - (Bank Template)

Current Portion of Long-Term Debt - (Bank Template)

Current Portion of Leases

Long-Term Debt

Federal Home Loan Bank Debt - LT

Long Term Leases

Trust Pref. Securities

All other templates:

Short-Term Borrowings - (Template Specific)

Curr. Port. of LT Debt

Current Portion of Leases

Long-Term Debt

Long Term Leases

Trust Pref. Securities

```python
# Among Honeywell, Caterpillar, Pfizer, PepsiCo, and Boeing,
# What is the Total Revenues of the company that has the lowest Total Debt?

from functions import (
    select_documents,
    retrieve_relevant_pages,
    extract_value
)

# Step 1: Identify the total debt for each company
companies = ["Honeywell", "Caterpillar", "Pfizer", "PepsiCo", "Boeing"]
# Assuming these are the correct stock symbols
stock_symbols = ["HON", "CAT", "PFE", "PEP", "BA"]
total_debts = {}

for company, symbol in zip(companies, stock_symbols):
    question_debt = f"What is the total debt of {company} ({symbol}) in 2023 in US
        dollars?"
    documents = select_documents(stock_symbols=[symbol], form_types=["10-K"],
        fiscal_years=[2023])
    pages = retrieve_relevant_pages(question_debt, documents)
    total_debt = extract_value(question_debt, pages)
total_debts[company] = total_debt

# Step 2: Determine which company has the lowest total debt
lowest_debt_company = min(total_debts, key=total_debts.get)

# Step 3: Find the total revenues of that company
question_revenue = f"What is the total revenues of {lowest_debt_company} in 2023 in
        US dollars?"
symbol = stock_symbols[companies.index(lowest_debt_company)]
documents = select_documents(stock_symbols=[symbol], form_types=["10-K"],
    fiscal_years=[2023])
pages = retrieve_relevant_pages(question_revenue, documents)
total_revenues = extract_value(question_revenue, pages)

# Save the answer to a file
with open("answer.txt", "w") as f:
    f.write(total_revenues)
```

Figure 7: An example of code generated by GPT-4 in the CodeGen+DocS+PageR system. The LLM models successfully decompose the question into two main steps: (1) determine the company with the lowest total debt and (2) extract the Total Revenues of that company. In the first step, the question is further decomposed into atomic questions for the Document Selection task and the Page Retrieval task. This fine-grain process ensures the higher accuracy of the system compared to the coarse-grain systems such as Vanilla RAG.

```
1  You are a financial expert.
2  The most current fiscal year is {current_year}
3  You can answer quantitative finance questions by writing Python code using helpful
       functions.
4  There are two functions:
5  - select_document: return a list of supporting documents.
6  - retrieve_relevant_pages: return a list of relevant pages that contain information
        to answer the question from the list of documents
7  - extract value: return an extracted value from the given document
8  select_document(
9      companies: list = None,
10     stock_symbols: list = None,
11     form_types: list = None,
12     fiscal_years: list = None,
13     financial_period_end_date_range_start: str = None,
14     financial_period_end_date_range_end: str = None
15 ):
16     """
17     This function matches documents by a series of conditions.
18     If the condition is not empty, they must match all given condition
19     companies and stock_symbols are not mutually exclusive. A document is matched if
        satisfies one of these conditions.
20     The documents must belong to one of the companies specified by the companies or
        stock_symbols
21     The financial period end date to filter must be between (
        financial_period_end_date_range_start, financial_period_end_date_range_end)
22     :param companies: a list of a few desired company short names.
23     :param stock_symbols: a list of the corresponding companies' stock ticker symbols
24     :param form_types: a list of the form types such as "8-K" for the current report,
        "10-K" for the annual report, "10-Q" for the quarterly report
25     :param fiscal_years: a list of the corresponding companies' fiscal years
26     :param financial_period_end_date_range_start: the beginning of a range used to
        filter financial period end date in "yyyy-mm-dd" format
27     :param financial_period_end_date_range_end: the beginning of a range used to
        filter financial period end date in "yyyy-mm-dd" format
28
29     :return: a list of supported documents. Return an empty list [] if no document is
         matched.
30     """
31
32 retrieve_relevant_pages(question: str, documents: list):
33     """
34     :param question: a financial question
35     :param documents: a list of documents, each with multiple pages
36     :return: (list of str)a short list of pages
37     """
38
39 extract_value(
40    question: str,
41    pages: list
42 ):
43     """
44     :param qa: a question
45     :param pages: a list of pages
46     :return: an extracted value from the given list of pages.
47     If the value is a money amount, the returned value is a float number in US
        dollars.
48     If the value is a count, the returned value is a simple float number.
49     If the value is a percentage, the returned value is a float number. E.g., 1%
        would be returned as 0.01.
50     If the question is a yes-no question, it would return "Yes" or "No" only.
51     """
52 Finally, you must write the short answer to a file named "answer.txt". The answer
       must be short, just a Yes/No, or a number
```

Figure 8: The prompt for CodeGen+DocS+PageR

```
1  Here are some examples:
2  Question: Did Coca-Cola pay dividends in 2017?
3  Python Code:
4  from functions import select_document
5  from functions import retrieve_relevant_pages
6  from functions import extract_value
7  documents = select_document(stock_symbols=["KO"], form_types=["10-K"], fiscal_years
       =[2017])
8  question = "How much did Coca-Cola pay dividends in USD in 2017?"
9  pages = retrieve_relevant_pages(question, documents)
10 value = extract_value(question, pages)
11 if isinstance(value, str):
12     if value == "yes":
13         answer = "Yes"
14     elif value == "no":
15         answer = "No"
16     else:
17         dividends = float(value)
18 else:
19     dividends = float(value)
20 if dividends > 0:
21     answer = "Yes"
22 else:
23     answer = "No"
24 with open("answer.txt", "w") as f:
25     f.write(answer)
26
27 Question: What is the overall revenue growth of Abbott over the last 2-year period?
28 Python Code:
29 from functions import select_document
30 from functions import retrieve_relevant_pages
31 from functions import extract_value
32 current_year = {current_year}
33 question = f"How much did Coca-Cola pay dividends in {{current_year}} in USD?"
34 documents = select_document(stock_symbols=["KO"], form_types=["10-K"], fiscal_years
       =[current_year])
35 pages = retrieve_relevant_pages(question, documents)
36 value_current = extract_value(question, pages=pages)
37 base_year = current_year - 2
38 question = f"How much did Coca-Cola pay dividends in {{base_year}} in USD?"
39 documents = select_document(stock_symbols=["KO"], form_types=["10-K"], fiscal_years
       =[base_year])
40 pages = retrieve_relevant_pages(question, documents)
41 value_base = extract_value(question, pages)
42 growth_percentage = (value_current - value_base) / value_base * 100.0
43 with open("answer.txt", "w") as f:
44     f.write(str(growth_percentage))
45
46 Question: How much did NFLX return to the investors in the last 3 years?
47 Python Code:
48 from functions import select_document
49 from functions import retrieve_relevant_pages
50 from functions import extract_value
51 current_year = {current_year}
52 returned_values = []
53 for year in range(current_year, current_year - 3, -1):
54 question = f"How much did Netflix return to the investors in {{year}} in USD?"
55     documents = select_document(companies=["Netflix"], stock_symbols=["NFLX"],
       form_types=["10-K"], fiscal_years=[year])
56     pages = retrieve_relevant_pages(question, documents)
57     return_in_us_dollars = extract_value(question, pages)
58     returned_values.append(float(return_in_us_dollars))
59 total_return = sum(returned_values)
60 with open("answer.txt", "w") as f:
61     f.write(str(total_return))
```

Figure 9: The few-shot demonstrations used for CodeGen+DocS+PageR system.