A Workflow for Financial Analysis using LLM and OFIN: A Benchmark for Evaluating LLM Capabilities in Financial Matrix Calculation

Haonan Chen hc3441

hc3441@columbia.edu

Abstract

This thesis presents a six-step end-to-end workflow designed to calculate financial matrices using generative AI and establishes a benchmark to evaluate various models' performance in this task. The findings indicate that the maximum overall accuracy rate is 31.25%, which is lower than initially anticipated. However, when a tolerance of ±10% is applied, the maximum overall accuracy rate improves to 71.25%. Furthermore, the study identifies universal strengths and weaknesses across all models. The observed limitations are attributed to factors such as the requirement for cross-period data, ambiguity in metric definitions, and the complexity of multi-step calculations. The experience gained from developing this workflow and benchmark can be applied to both further research and industry practices, providing valuable insights for improving AI performance in financial analysis tasks.

1. Introduction

With OpenAI's O1 model released in December 2024, achieving state-of-the-art (SOTA) performance on numerous benchmark sets such as MMLU and even PhD-level questions, there is a growing perception that generative AI, or Large Language Models (LLMs), have attained remarkable capabilities across industries rich in textual and numerical data. The finance industry, where professionals routinely handle vast amounts of structured and unstructured data, formatted and unformatted files, emerges as a particularly suitable domain for LLM-based applications.

Prominent fintech companies like Bloomberg, East Money, and other bulge-bracket firms are investing heavily in developing proprietary models to better serve their clients. In particular, sell-side equity researchers, who must analyze extensive information, including news, reports, and financial statements, to update their financial models, stand at the intersection of opportunity and uncertainty. On one hand, they anticipate leveraging powerful AI tools to enhance efficiency; on the other hand, they fear potential displacement by these advanced AI agents.

Two critical questions arise in the intersection of AI and financial analysis. The first concerns how to design an effective and cost-efficient workflow to integrate LLMs into financial analysis tasks. The second involves evaluating the capabilities of LLMs in this domain: how can we define and measure their performance, and how do we determine the most suitable model for a specific task? In essence, a well-defined workflow and a robust benchmark tailored for financial analysis are indispensable to addressing these questions.

Previous studies have already made significant endeavors in these areas, with most papers focusing on the benchmark side. Nie, Yan, Guo et al. (2024) created a benchmark to assess LLMs' capabilities in the Chinese finance sector, comprising 99,100 questions across four categories: financial subjects, financial qualifications, financial practices, and financial law. GPT-4 achieved the highest accuracy in this benchmark, scoring 60.16%. Similarly, Islam et al. (2023) constructed a test suite to evaluate LLMs' performance in answering financial questions. They tested 16 SOTA models using 150 cases derived from a total of 10,231 questions in the FinanceBench dataset and identified clear limitations across models, such as hallucinations.

Expanding on these efforts, Xie et al. (2024) introduced an extensive open-source evaluation benchmark spanning 24 financial tasks, including information extraction, textual analysis, question answering, text generation, and more. They found that while LLMs excelled in information extraction (IE) and textual analysis, they struggled with advanced reasoning and complex tasks like text generation and forecasting. GPT-4 performed best in IE and stock trading tasks, while Gemini outperformed in text generation and forecasting.

Further, Liu et al. (2024) assessed LLMs across three dimensions: foundational ability (e.g., financial calculations and risk assessment), reasoning ability (e.g., comprehending textual information and analyzing abnormal financial reports), and technical skills (e.g., data analysis and visualization). GPT-4, with 3-shot examples, ranked highest among the evaluated models.

In another study, Xie et al. (2023) introduced PIXIU, a dataset and benchmark designed for both fine-tuning and evaluation. They also proposed FinMA, a financial LLM developed by fine-tuning LLaMA. Finally, Srivastava et al. (2024) explored LLMs' mathematical reasoning abilities on four financial datasets: TATQA, FinQA, ConvFinQA, and MultiHiertt, using various prompt techniques, including Direct, Chain-of-Thought, Program-of-Thought, Decomposers, and Elicit-Extract-Decompose-Predict. They concluded that GPT-4 consistently dominated most benchmarks across nearly all prompting techniques.

Few studies have addressed the prompt or workflow perspective in financial analysis using LLMs. Nie, Kong et al. (2024) explored the application of LLMs across various financial tasks, emphasizing their potential in contextual understanding, transfer learning flexibility, and complex emotion detection. Kim, Muhn & Nikolaev (2024b) investigated whether LLMs can outperform human financial analysts in predicting earnings changes using standardized and anonymized financial statements. The results indicate that LLMs can indeed outperform humans, particularly in complex situations. Moreover, trading strategies based on GPT's predictions achieved a higher Sharpe ratio compared to other models.

Kim, Muhn & Nikolaev (2024a) examined attention-based models for identifying content that triggers market reactions, enabling the analysis of a wide range of topics, from goodwill to income. In a related work, Kim, Muhn, Nikolaev & Zhang (2024) explored the usefulness of LLMs in summarizing complex corporate disclosures in the stock market. They found that generative AI-based summaries are highly informative to investors, as they exclude irrelevant information and help reduce informational frictions.

In order to contribute further to the development of workflows while designing a benchmark for evaluating financial analysis abilities, this research introduces a six-step end-to-end workflow for financial matrix calculation using LLMs. By *financial matrix*, the author refers to ratios calculated based on data extracted from financial statements. These matrices typically represent a company's financial and operational characteristics to some extent and serve as valuable tools for understanding a company's performance. They are an essential and irreplaceable component in equity research and investment analysis.

Second, many of these ratios play a crucial role in various valuation methods, such as Return on Equity (ROE) and Earnings Per Share (EPS), which are key metrics in multiples-based valuation approaches. Additionally, numerous quantitative trading strategies are constructed based on these matrices.

In this work, the author specifically focuses on operating-related financial matrices—ratios that reflect the business quality of a company and are directly influenced by its operational activities. These matrices offer a clear perspective on how well a company manages its core business operations, making them an excellent starting point for value investing analysis. By emphasizing the main business activities, these ratios help avoid distortions caused by a company's financing or investment decisions.

This workflow includes table extraction, financial statement classification, time and subject labeling, formula generation, value extraction, and matrix calculation. The proposed pipeline effectively addresses existing challenges such as limited context window length, response latency, text input format, and token cost.

Second, the author constructs a benchmark dataset named OFIN, which consists of figures for 18 operating-related financial matrices from 5,442 Chinese listed companies for the fiscal year 2023, collected from the CSMAR database. Instead of building traditional question-answer pairs, the author tests several models from OpenAI and Anthropic using the developed workflow and compares their results to the OFIN benchmark to evaluate accuracy. The results show that Claude-3.5 Sonnet performs the best in this task, achieving an exact match accuracy of 31.25%, which increases to 71.25% when allowing for a ±10% tolerance. Additionally, the author observes that within the same model series, larger models consistently outperform their smaller counterparts.

Moreover, the author identifies universal patterns across different models. Almost every model excels at calculating the *Current Assets Revenue Ratio*, *Accounts Receivable Revenue Ratio*, and *Inventory Revenue Ratio*. In contrast, most models exhibit low accuracy when computing the *Accounts Payable Turnover*, *Inventory Days*, and *Cash Turnover*. Upon analyzing the formulas for these ratios, the author concludes that LLMs perform better when calculations require single-period data and single-step operations. Conversely, models struggle with ratios involving average figures, which demand cross-period data and multi-step calculations.

In summary, this paper introduces a novel workflow tailored for calculating financial matrices, constructs the OFIN benchmark for evaluating LLM performance in operating-related financial matrix calculations, evaluates state-of-the-art models using this benchmark, and identifies universal patterns across the models. This work not only contributes valuable experience in developing generative AI applications in the financial domain but also provides a robust tool for assessing model readiness. Furthermore, it generates critical insights into the strengths and limitations of LLMs in handling financial matrix calculations.

2. Research Method: End-to-End Pipeline for Financial Matrix Calculation

2.1 Table Extraction

An intuitive approach might be to feed the entire annual report into an API and request it to calculate the targeted metrics, similar to how one would interact with ChatGPT. However, in practice, this method has several significant drawbacks, including high token costs, API response latency, exceeding the API's context length limits, and overall inefficiency.

To address these challenges, this thesis employs a PDF table extraction Python library called `Camelot`. This tool extracts all tables from the annual reports, including the three consolidated financial statements required for analysis. This step prepares the data for subsequent precise table classification and targeted extraction.

2.2 Financial Statement Classification

An annual report typically contains approximately 300 to 400 tables. Feeding all of these into an API would still present issues such as high token costs, context length limits, and inefficiencies. Therefore, the next goal is to extract the three main consolidated financial statements, which are sufficient for calculating all the desired matrices.

A straightforward approach might involve searching for titles, such as "Consolidated Balance Sheet." However, in practice, the author observed that table titles are sometimes missing or not embedded directly within the tables. To overcome this challenge, the author implemented a more robust approach: defining a set of key subjects. If a table contains three or more key subjects, it is identified as relevant and saved. Additionally, tables immediately preceding or following the identified table are also saved, as consolidated financial statements often span multiple pages and are sometimes recognized as separate tables by `Camelot`.

The criteria for selecting key subjects were that they should be commonly found in the target statement but rarely appear in the other two types of financial statements. Below are examples of key subjects used to identify each type of financial statement:

```
 key_subjects = {
  "Balance Sheet":  ["Total Assets", "Total Liabilities", "Total
Equity"...],
  "Income Statement":  ["Revenue", "Cost of Goods Sold", "Taxes"...],
  "Cash Flow Statement":  ["Operating Activities", "Investment
Activities", "Financing Activities"...]  }
```

2.3 Time and Subject Labeling

After extracting the three main financial statements, an additional step is required before feeding the data into the API. Since the API can only accept inputs in text format, it is necessary to convert all tables into strings. However, the default logic for this conversion in computers processes each row sequentially and concatenates them into a single string. This approach causes date information, typically presented as column headers, to appear only once at the beginning of the text. This creates significant challenges when extracting cross-period values and calculating matrices that require average data.

To address this issue, the author developed a method that processes each column of the table individually. Specifically, each column is fed into the API, and if the API detects time-related information in the column, the time label is appended to the corresponding data. The same operation is applied to accounting subjects for similar reasons, as column-based data must retain subject context to ensure accurate extraction and calculation.

The prompt used for this step is as follows:

```
 prompt = """Analyze and label the time and subject information of the
financial table.
Table Content:  {df.to_string()}
Instructions:
1.  Determine if this is a valid financial table (containing accounting
subjects and corresponding values).
2.  Label the time (e.g., 2023 Annual, End of 2022) and subjects.
3.  Clean numerical values:
- Remove parentheses, percentages, etc., and retain pure numerical
values.
- cleaned_value must be a valid number, and cannot be an empty string.
- If a cell does not contain numerical values, do not include it in
cells_to_label.
- For headers, subtotals, or totals, do not include these rows in the
results."""
```

2.4 Formula Generation

In this step, the model is tasked with generating formulas for each financial matrix. These formulas are then used to identify and collect all the accounting subjects along with the relevant time periods required for extraction.

The author employed few-shot prompting techniques to guide the model in adhering to the desired return format of "time points + subject names". The prompt used for this step is as follows:

```
prompt = f"""For the following financial matrices:  {',
'.join(matrix_list)}
Return ONLY a list of all unique required time points + subjects across
all matrices.  For example:
   ["End of 2022 Accounts Receivable", "2023 Annual Net Profit", "End of
2023 Cash Balance"]
Do NOT include duplicates.  Each time point + subject combination should
appear only once."""
```

2.5 Value Extraction

In this step, the labeled table and the previously generated formula components are both fed into the API. The API is tasked with extracting the required numerical values based on the given formula and table structure.

There are a few noteworthy techniques applied during this process. First, since the original file is in Chinese, different names can refer to the same subject. To address this, the author instructed the AI to account for such variations. Second, it is possible that multiple values exist for the same subject within different contexts. For example, Accounts Receivable may appear both for individual customers and as a total. Instead of halting the extraction process immediately upon encountering such ambiguity, the author asked the AI to process all the values and determine the most appropriate one to use.

The following prompt was utilized for this step:

```
 prompt = f"""Extract financial report data.  Instructions:
1.  Handle special formats:  convert bracketed values to negative numbers
and percentages to decimals.
2.  Identify different terms:  e.g., "Total Equity" and "Shareholders'
Equity" are equivalent.
3.  If a subject appears multiple times, select the most reasonable value
based on context.
4.  Prioritize data from subtotal/total rows.
Subjects to Extract:  {formatted_subjects}
Financial Report Data:
{chr(10).join(f"{category}:{chr(10){chr(10).join(tables)}" for category, tables in
all_tables.items())}"""
```

2.6 Matrix Calculation

Finally, to calculate the matrices, all the extracted data and the names of the matrices were fed into the API, and the model was tasked with performing the necessary calculations. To effectively track errors without impacting the accuracy rate calculation, error codes were introduced to represent different types of potential errors.

The prompt used for this step is as follows:

```
prompt = f"""Calculate the metrics in the given list using the provided
values.
Return format: {Metric Name: Value}
If a calculation error occurs, return the corresponding error code as the
value based on the following rules:
- Missing data required for calculation: -1001
- Division by zero: -1002
- Invalid calculation result (e.g., negative where not allowed): -1003
- Data format error: -1004
- Other calculation errors: -1005
Example:
{
   "Current Ratio": 1.5,
   "Asset Turnover": -1001, # Missing data
   "Gross Profit Margin": -1002 # Division by zero
}
"""
```

3. Data Resources and Results

3.1 Data Resources and Descriptive Statistics

The financial matrices data used as the answer set for the benchmark part comes from the *CSMAR* database, a widely recognized and frequently used resource that provides comprehensive financial information for most Chinese listed companies. The annual reports were collected from *coinfo.com*, an official information disclosure platform selected by the China SEC. This study gathered the annual reports of **5,442 companies** for the year **2023**, covering over **99%** of all Chinese A-share listed companies.

3.2 Selected Matrices and Rationales

The author selected **18 financial matrices**, all of which are *operating-related* and can serve as indicators to assess the business quality of a company to a certain extent. These matrices were chosen for their ability to reflect different aspects of a company's financial performance and operational efficiency.

The selected matrices are divided into **5 categories** as follows:

Table 1: Selected Matrices and Their Business Significance

Category	Description	Key Metrics
Collection & Payment Management Efficiency	Measures the working capital cycle through receivables and payables, ensuring operational liquidity.	Accounts Receivable Revenue Ratio, Accounts Payable Turnover
Inventory Management Efficiency	Evaluates inventory handling efficiency, providing insights into operational effectiveness.	Inventory Revenue Ratio, Inventory Days
Working Capital Efficiency	Assesses short-term operational efficiency and liquidity management by analyzing turnover and cycle duration.	Working Capital Turnover, Operating Cycle
Asset Utilization Efficiency	Examines how effectively a company utilizes its assets to generate revenue.	Current Assets Turnover, Total Assets Turnover
Capital Structure & Efficiency	Focuses on the efficiency of capital investment and the balance of equity utilization.	Capital Intensity, Equity Turnover

Each of these categories and their corresponding matrices provide critical insights into specific dimensions of a company's operations, enabling a holistic evaluation of business performance and operational quality.

The descriptive statistics of the data are summarized in the following table:

Table 2: Descriptive Statistics of Financial Matrices

Matrix	Missing Values	Mean	Variance
Total Assets Turnover	0	0.5679	0.2215
Fixed Assets Turnover	0	23.5970	320395.96
Cash Turnover	0	6.2126	173.29
Inventory Revenue Ratio	2	0.3498	6.3504
Fixed Assets Revenue Ratio	2	0.6345	32.5885
Capital Intensity	2	9.5243	98725.85
Accounts Receivable Revenue Ratio	2	0.5194	237.8375
Equity Turnover	20	1.3328	11.8450
Operating Cycle	48	477.4370	70758362.28
Accounts Receivable Turnover	60	152.3411	29472875.14
Accounts Receivable Days	62	177.7930	22094210.18
Noncurrent Assets Turnover	98	2.2311	28.7459
Current Assets Turnover	98	1.0934	0.8611
Current Assets Revenue Ratio	99	6.1963	60335.77
Accounts Payable Turnover	103	13.5178	140594.97
Inventory Turnover	158	384.8570	189358774.95
Inventory Days	158	306.3529	49771184.42
Working Capital Turnover	841	5.7922	1874.4018

3.3 Overall Evaluation Results

The evaluation results of different models from OpenAI and Anthropic on the benchmark are summarized in Table 3, which includes exact match accuracy rates and results with tolerances of ±1%, ±5%, and ±10%:

Table 3: Performance Evaluation of LLM Models

Model	**Exact Match**	**Within ±1%**	**Within ±5%**	**Within ±10%**
Claude-3.5 Sonnet	31.25%	50.00%	57.50%	71.25%
OpenAI O1 Preview	27.78%	38.89%	44.44%	55.56%
OpenAI O1 Mini	20.00%	20.00%	26.25%	41.25%
Claude-3.5 Haiku	15.00%	15.00%	25.00%	40.00%
OpenAI GPT-4o	15.00%	15.00%	20.00%	30.00%
OpenAI GPT-4o Mini	6.25%	6.25%	10.00%	16.25%

The first noteworthy observation is that no model achieved an exact match accuracy rate exceeding 32%, raising concerns about the practical applicability of generative AI in tasks like this. Claude-3.5 Sonnet performed best, with an exact match accuracy rate of 31.25% and an accuracy of 71.25% within a 10% tolerance. O1 Preview ranked second, achieving 27.78% and 55.56%, respectively.

Another interesting finding is that within the same model series, larger models consistently outperformed their smaller counterparts. For example, Sonnet outperformed Haiku by 16.25%, O1 Preview surpassed O1 Mini by 7.78%, and GPT-4o exceeded GPT-4o Mini by 8.75%. This trend suggests that as model size increases, their performance on financial matrix tasks may improve, indicating a potential for stronger results with further scaling of models.

3.4 Universal Strengths and Weaknesses

Beyond the overall accuracy, the author further investigated whether certain matrices consistently exhibit strong or poor performance across all models. In Table 4, the matrices are ranked based on their *average accuracy rank* across models.

The results reveal that some matrices, such as *Current Assets Revenue Ratio*, *Accounts Receivable Revenue Ratio*, and *Inventory Revenue Ratio*, achieve consistently high accuracy, averaging among the top 5. Conversely, matrices like *Accounts Payable Turnover*, *Inventory Days* and *Cash Turnover*, consistently rank among the bottom 5 in terms of accuracy. This trend is further visualized in the heatmap presented in Figure 1, which highlights the varying levels of performance across matrices.

Table 4: Average Rank of Financial Matrices Across Models

Matrix	**Average Rank**
Current Assets Revenue Ratio	2.83
Accounts Receivable Revenue Ratio	3.50
Inventory Revenue Ratio	4.50
Current Assets Turnover	5.83
Total Assets Turnover	6.50
Fixed Assets Revenue Ratio	7.83
Accounts Receivable Turnover	8.67
Fixed Assets Turnover	8.83
Capital Intensity	9.17
Equity Turnover	9.50
Noncurrent Assets Turnover	10.67
Accounts Receivable Days	10.83
Operating Cycle	12.17
Working Capital Turnover	12.67
Inventory Turnover	13.67
Accounts Payable Turnover	14.50
Inventory Days	14.67
Cash Turnover	14.67

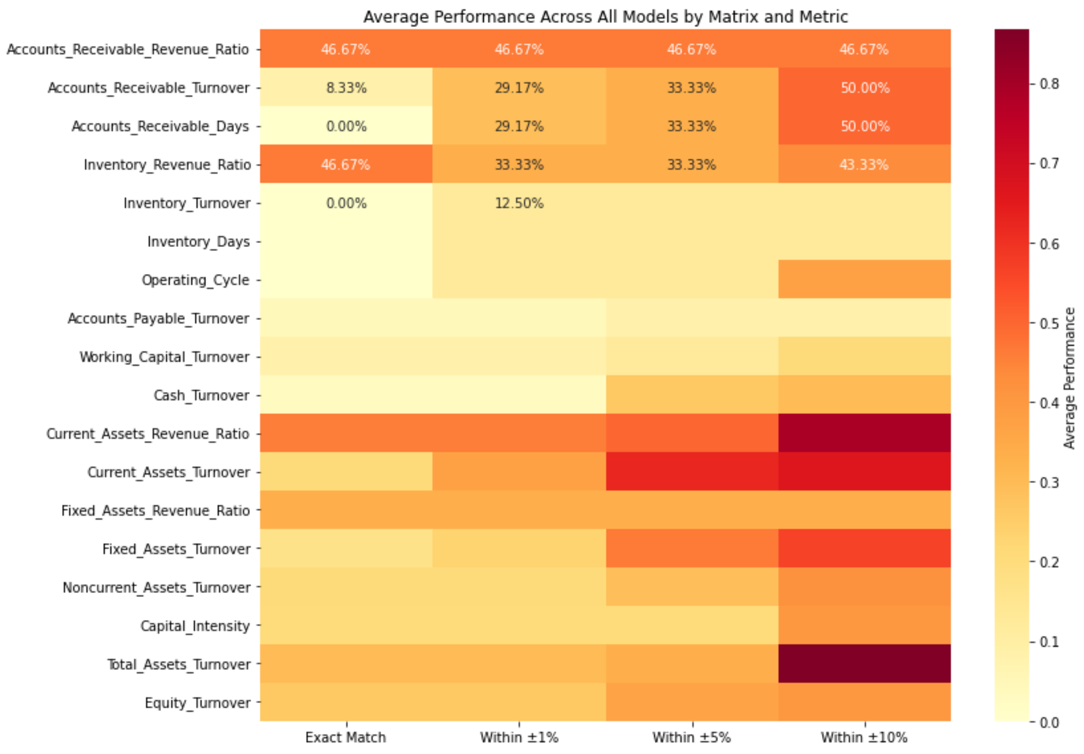

Figure 1: Average Performance Across All Models by Matrix and Metric

To further hypothesize the reasons behind the observed universal strengths and weaknesses, we analyze the formulas of six financial matrices in Table 5. The table highlights the strengths and weaknesses of each formula.

Table 5: Comparison of Strengths and Weaknesses in Financial Matrices

Strengths	$\text{Current Assets Revenue Ratio} = \dfrac{\text{Current Assets}}{\text{Revenue}}$
	$\text{Accounts Receivable Revenue Ratio} = \dfrac{\text{Accounts Receivable}}{\text{Revenue}}$
	$\text{Inventory Revenue Ratio} = \dfrac{\text{Inventory}}{\text{Revenue}}$
Weaknesses	$\text{Accounts Payable Turnover} = \dfrac{\text{COGS}}{\text{Average Accounts Payable}}$
	$\text{Inventory Days} = \dfrac{\text{Average Inventory}}{\text{COGS}} \times 365$
	$\text{Cash Turnover} = \dfrac{\text{Revenue}}{\text{Average Cash}}$

As seen in Table 5, it is not difficult to observe that the first three formulas—*Current Assets Revenue Ratio*, *Accounts Receivable Revenue Ratio*, and *Inventory Revenue Ratio*—share three common characteristics. First, each formula requires only one straightforward computation, making them single-step calculations. Second, the data used in these formulas are restricted to the current period, avoiding the need for cross-period references. Third, all three formulas include the *Revenue* subject, which is universally present across all companies' financial statements.

In contrast, the last three formulas—*Accounts Payable Turnover*, *Inventory Days*, and *Cash Turnover*—exhibit distinct complexities. These formulas include average terms, which necessitate data across multiple time periods, introducing a cross-period data requirement. Furthermore, additional steps, such as calculating averages and incorporating terms like *COGS* or *Average Inventory*, increase the computational complexity and make these formulas multi-step in nature.

These distinctions likely explain why the first three matrices consistently achieve higher accuracy across models, while the latter three tend to perform poorly due to their reliance on cross-period data and multi-step calculations.

4. Conclusion

In this paper, the author presents an end-to-end solution for calculating financial matrices using Generative AI APIs. The proposed solution consists of six systematic steps: table extraction, financial statement classification, time and subject labeling, formula generation, value extraction, and matrix calculation. This workflow effectively addresses challenges such as limited context window length, high token cost, loss of time information, and API response latency.

Using this workflow, the author constructs an operating-related financial matrices benchmark answer set, termed *OFIN*. The annual report data originates from cninfo.com, while the financial matrices data is sourced from CSMAR. Leveraging this benchmark, the author evaluates advanced models provided by OpenAI and Anthropic, observing a highest exact match accuracy of 31.25% and a ±10% tolerance accuracy of 71.25%. Results across different models also highlight a clear trend: within the same model series, larger models consistently achieve higher accuracy.

Furthermore, the author identifies patterns across models, revealing that LLM performs notably better when tasks require single-step calculations and rely solely on single-period data. Conversely, performance deteriorates significantly when average terms and multi-step calculations are involved.

In summary, this paper provides practical insights into building AI workflows for finance and accounting, introduces a robust and user-friendly benchmark for testing model capabilities in this domain, and demonstrates the strengths and limitations of state-of-the-art models. These findings offer valuable experience for future research, paving the way for further exploration of AI applications in financial statement analysis.

Data Availability

The financial matrices data, sourced from the operational matrices library of *CSMAR*, is available at the following site: https://data.csmar.com/. The annual reports used in this study can be accessed and downloaded from the author's GitHub repository: https://github.com/HenryChen404?tab=repositories.

Acknowledgments

I extend my heartfelt gratitude to Professor Thomas Bourveau for his sincere help and invaluable guidance throughout this project. I would also like to express my deep appreciation to Professor Tianyi Peng and Hannah Li for organizing the course *Generative AI: Technical and Social*, where I acquired the techniques applied in this project and gained the confidence to pursue generative AI-related research.

Special thanks to Professor Stephen Penman for leading the *MSAFA* program, which provided me with the opportunity to study at Columbia Business School. This experience has inspired my career aspirations, introduced me to lifelong friends, and created cherished memories over a beautiful one and a half years that I will treasure for the rest of my life.

References

Islam, P., Kannappan, A., Kiela, D., et al. (2023). FINANCEBENCH: A new benchmark for financial question answering. *arXiv preprint*, 2311(11944).

Kim, A. G., Muhn, M., & Nikolaev, V. V. (2024a). Bloated disclosures: Can ChatGPT help investors process information? *arXiv preprint*, 2306(10224).

Kim, A. G., Muhn, M., & Nikolaev, V. V. (2024b). Financial statement analysis with large language models. *arXiv preprint*, 2407(17866).

Kim, A. G., Muhn, M., Nikolaev, V. V., & Zhang, Y. (2024). Learning fundamentals from text. *Chicago Booth Accounting Research Center Research Paper*, 2024-155(5047947).

Liu, S., Zhao, S., Jia, C., et al. (2024). FinDaBench: Benchmarking financial data analysis ability of large language models. *arXiv preprint*, 2401(02982).

Nie, Y., Yan, B., Guo, T., et al. (2024). The CfinBench: A comprehensive chinese financial benchmark for large language models. *arXiv preprint*, 2407(02301).

Nie, Y., Kong, Y., Dong, X., Mulvey, J. M., Poor, H. V., Wen, Q., & Zohren, S. (2024). A survey of large language models for financial applications: Progress, prospects and challenges. *arXiv preprint*, 2406(11903).

Srivastava, P., Malik, M., Gupta, V., Ganu, T., & Roth, D. (2024). Evaluating LLMs' mathematical reasoning in financial document question answering. *arXiv preprint*, 2402(11194).

Xie, Q., Han, W., Chen, Z., et al. (2024). FinBen: A holistic financial benchmark for large language models. *arXiv preprint*, 2402(12659).

Xie, Q., Han, W., Zhang, X., et al. (2023). PIXIU: A large language model, instruction data and evaluation benchmark for finance. *arXiv preprint*, 2306(05443).

Appendix

A. Code

A.1 Setup and Libraries

The following Python code imports necessary libraries, sets up logging, and initializes API keys for different LLM providers. The libraries include `pandas`, `camelot`, and custom AI clients for data extraction.

```python
import pandas as pd
import glob
from pathlib import Path
import camelot
import json
import aisuite as ai
import os
from typing import Dict, List, Tuple, Any
import logging
import sys

# Set up logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.StreamHandler(sys.stdout),
        logging.FileHandler('financial_analysis.log')
    ]
)

logger = logging.getLogger(__name__)

# Set API keys as environment variables
os.environ['OPENAI_API_KEY'] = "YOUR_API_KEY"
os.environ['ANTHROPIC_API_KEY'] = "YOUR_API_KEY"
os.environ['GOOGLE_API_KEY'] = "YOUR_API_KEY"
os.environ['MISTRAL_API_KEY'] = "YOUR_API_KEY"

# Initialize aisuite client
client = ai.Client()
```

Listing 1: Python Code for Logging Setup

A.2 Read and Clean Excel Data

The code reads the input Excel file, filters data for specific conditions, and renames columns using a predefined mapping.

```python
# Read an excel file
ori_df = pd.read_excel(YOUR_ANSWER_SET_EXCEL)

# Drop initial rows and reset index
df = ori_df.drop([0, 1]).reset_index(drop=True)
df = df[df['Accper'] == "2023-12-31"]
df = df[df['Typrep'] == "A"]
df = df[df['Source'] == 0]

print(df)
```

```python

# Create renaming dictionary
rename_dict = {
    'F040101B': 'Accounts_Receivable_Revenue_Ratio',
    'F040202B': 'Accounts_Receivable_Turnover',
    'F040302B': 'Accounts_Receivable_Days',
    'F040401B': 'Inventory_Revenue_Ratio',
    'F040502B': 'Inventory_Turnover',
    'F040602B': 'Inventory_Days',
    'F040702B': 'Operating_Cycle',
    'F040802B': 'Accounts_Payable_Turnover',
    'F040902B': 'Working_Capital_Turnover',
    'F041002B': 'Cash_Turnover',
    'F041101B': 'Current_Assets_Revenue_Ratio',
    'F041202B': 'Current_Assets_Turnover',
    'F041301B': 'Fixed_Assets_Revenue_Ratio',
    'F041402B': 'Fixed_Assets_Turnover',
    'F041502B': 'Noncurrent_Assets_Turnover',
    'F041601B': 'Capital_Intensity',
    'F041702B': 'Total_Assets_Turnover',
    'F041802B': 'Equity_Turnover'
}

# Rename DataFrame columns
df = df.rename(columns=rename_dict)

# Randomly sample 6 rows
df1 = df.sample(n=6, random_state=101)

# Replace missing values and round
matrix = [*rename_dict.values()]
df1 = df1.fillna(-101)
df1 = df1.round(2)

# Delete specified columns
columns_to_drop = ['ShortName', 'Accper', 'Typrep', 'Indnme1',
   'Source']
df1 = df1.drop(columns=columns_to_drop)
```

Listing 2: Rename and Process DataFrame in Python

A.3 Descriptive Statistics

The following function calculates basic descriptive statistics, such as missing values, means, and variances for each column in the DataFrame.

```python
def descriptive_statistics(df):
    stats = pd.DataFrame(\{
        "Missing Values": df.isnull().sum(),
        "Mean": df.mean(),
         "Variance": df.var()
    })
    return stats

# Execute function
df = df.drop(columns=column_to_drop)
df = df.drop(columns=['Stkcd'])
df = df.apply(pd.to\_numeric, errors='coerce')

result = descriptive\_statistics(df)
print(result)
```

Listing 3: Descriptive Statistics

A.4 File Handling and JSON Cleaning

The following functions handle file searches for financial reports and clean AI-generated JSON responses.

```python
def check_files(Stkcd, pdf_folder):
    """Check if PDF file exists for a given stock code."""
    stkcd = str(Stkcd).zfill(6)
    pattern = str(Path(pdf_folder) / f"{stkcd}*.pdf")
    matching_files = glob.glob(pattern)

    if matching_files:
        print(f"Found file for {stkcd}: {matching_files[0]}")
        return matching_files[0]
    else:
        print(f"No file found for {stkcd}")
        return False

def clean_json_response(content):
    """Clean and format JSON response."""
    try:
        json_start = content.find("```json") + 7
        json_end = content.find("```", json_start)

        if json_start > 6 and json_end > json_start:
            json_content = content[json_start:json_end].strip()
            return json_content
        else:
            start = content.find('{')
            end = content.rfind('}') + 1
            if start >= 0 and end > start:
                return content[start:end]

        raise ValueError("No valid JSON content found.")

    except Exception as e:
        print(f"Error cleaning JSON: {str(e)}")
        print("Original content:", content)
        return content
```

Listing 4: Check Files and Clean JSON Response

A.5 Identifying Financial Statements

This function identifies relevant financial statements such as balance sheets, income statements, and cash flow statements by matching key subjects.

```python
def find_statements_by_subjects(tables):
    """Identify relevant financial statements using key subjects."""
    key_subjects = {
        "Balance Sheet Related": [
            "Total Assets", "Total Liabilities", "Total Equity",
            "Cash and Cash Equivalents", "Accounts Receivable",
            "Fixed Assets", "Deferred Tax Liabilities"
        ],
        "Income Statement Related": [
            "Operating Revenue", "Operating Cost", "Net Profit",
            "Taxes and Surcharges", "Operating Profit",
            "Income Tax Expense"
        ],
        "Cash Flow Statement Related": [
            "Operating Activities", "Investing Activities",
            "Financing Activities", "Cash and Cash Equivalents"
        ]
```

```python
    }

    categorized_tables = {
        "Balance Sheet Related": [],
        "Income Statement Related": [],
        "Cash Flow Statement Related": []
    }

    for i, table in enumerate(tables):
        df = table.df
        df_str = df.to_string().lower()

        for statement_type, subjects in key_subjects.items():
            matched_subjects = [
                subject for subject in subjects
                if subject.lower() in df_str
            ]

            if len(matched_subjects) >= 3:
                categorized_tables[statement_type].append(
                    {'table_index': i, 'matched_subjects':
                        matched_subjects}
                )

    return categorized_tables
```

Listing 5: Identify Financial Statements

A.6 Extracting Formulas and Values

Extract unique formulas and values from labeled tables using AI models.

```python
def get_all_formulas(model, matrix_list):
    """Extract unique required time points and subjects."""
    prompt = f"""
    For the following financial matrices: {', '.join(matrix_list)}
    Return a list of unique time points and subjects.
    Example: ["2022 Year-End Accounts Receivable", "2023 Annual
       Net Profit"]
    """

    response = client.chat.completions.create(
        model=model,
        messages=[
            {"role": "system", "content": "Financial expert, return only JSON format."},
            {"role": "user", "content": prompt}
        ],
        temperature=0.1
    )

    formula_info = json.loads(response.choices[0].message.content)
    return formula_info

def extract_all_values(model, labeled_tables, formula_info):
    """Extract values based on labeled tables and formulas."""
    prompt = f"""
    Extract financial data from the tables:
    Subjects: {', '.join(formula_info)}
    Return in JSON: {"Subject Name": Value}
    """

    response = client.chat.completions.create(
        model=model,
        messages=[
            {"role": "user", "content": prompt}
        ]
    )

    return json.loads(response.choices[0].message.content)
```

Listing 6: Extract Formulas and Values

A.7 Results and Accuracy

Calculate financial metrics and accuracy compared to ground truth data.

```python
def calculate_matrices(model, all_values, matrix_list):
    """Calculate financial metrics."""
    prompt = {"values": all_values, "metrics": matrix_list}

    response = client.chat.completions.create(
        model=model,
        messages=[{"role": "user", "content": json.dumps(prompt)}]
    )

    return json.loads(response.choices[0].message.content)
```

Listing 7: Calculate Financial Matrices

A.8 Main Execution

The main execution orchestrates the pipeline: extract tables, label data, calculate metrics, and save results.

```python
if __name__ == "__main__":
    pdf_folder = "YOUR_PDF_FOLDER"
    results_df = pd.DataFrame(columns=['Stkcd'] + matrix_list)

    formula_info = get_all_formulas(model, matrix_list)

    for stkcd, tables in tables_by_company.items():
        labeled_tables = add_labels_using_gpt(model, tables)
        all_values = extract_all_values(model, labeled_tables,
           formula_info)
        results = calculate_matrices(model, all_values,
           matrix_list)
        results_df = pd.concat([results_df,
           create_results_df(results, stkcd)])
```

Listing 8: Main Execution Pipeline