OFIN: A Benchmark for Evaluating LLM Capabilities in Financial Matrix Calculation

Haonan Chen hc3441

hc3441@columbia.edu

Abstract

In this work, the author presents OFIN, a benchmark designed to evaluate Large Language Models' (LLMs') capabilities in calculating operating-related financial matrices from annual reports. The author tested multiple LLM providers on 18 key operational metrics using data from 5,442 Chinese listed companies' 2023 annual reports. Through an end-to-end automated pipeline and validation against the CSMAR database, the results show that current LLMs have limited capabilities in this domain, with the best model achieving only 31.25% exact match accuracy. Furthermore, the author found that LLMs perform better on simple single-period calculations than on complex cross-period matrices. These findings provide insights for future development in this field.

1. Introduction

Large Language Models (LLMs) have shown remarkable success in natural language understanding and reasoning, but their capabilities in financial matrix calculation remain underexplored. Financial matrices, derived from structured data, such as annual reports, are critical to evaluating the operational efficiency and financial health of a company. Automating the extraction and calculation of these metrics can significantly improve accuracy and efficiency in financial analysis. However, current LLMs face challenges such as context window limitations, difficulty with structured tables, and errors in multistep numerical reasoning, particularly for multi-period calculations.

Existing research highlights both the potential and limitations of LLMs in finance. For example, Nie et al. (2024) propose **CFinBench**, a comprehensive Chinese financial benchmark for evaluating LLM capabilities. Similarly, benchmarks such as Xie et al. (2024) and Liu et al. (2024) emphasize the importance of holistic and task-specific financial evaluation.

From a mathematical reasoning perspective, Srivastava et al. (2024) demonstrate that LLMs struggle with complex arithmetic reasoning, particularly in financial documents. Furthermore, Kim et al. (2024) explore the application of LLMs in financial statement analysis, identifying gaps in their ability to process raw numerical data. These findings

are further supported by Islam et al. (2023), who highlight LLMs' difficulties in financial question answering, especially with nuanced and multi-step reasoning tasks.

Another critical benchmark, Liu et al. (2024), evaluates financial data analysis ability across various LLMs, shedding light on their strengths and weaknesses. Finally, **pixiu2024<empty citation>** introduce a large-scale instruction dataset tailored to financial tasks, showcasing the need for domain-specific data to improve LLM performance.

To address these limitations, the author introduces **OFIN**, a benchmark specifically designed to evaluate LLMs in extracting and calculating 18 key financial matrices from annual reports. Using data from more than 5,400 Chinese listed companies, OFIN provides a comprehensive assessment of LLM performance in financial data processing. By systematically analyzing their strengths and weaknesses, this project offers critical insights into improving LLMs for real-world financial applications.

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

2.1 Table Extraction

I implemented PDF table extraction using `Camelot` with stream flavor settings to handle the complex formatting of financial reports. The extraction process includes comprehensive error handling to manage various PDF formatting issues. To address potential character encoding problems, particularly with Chinese text, I incorporated UTF-8 encoding and suppressed standard output for cleaner processing.

2.2 Financial Statement Classification

The classification system identifies balance sheets, income statements, and cash flow statements through a key subject matching algorithm. I developed a scoring mechanism that evaluates tables based on the presence of characteristic accounting items, requiring a minimum of three matched subjects per statement type. The system also considers table dimensions and contextual relevance, managing both merged and split financial statements effectively.

The classification uses predefined key subjects for each statement type, for example:

```
 key_subjects = {
  "Balance Sheet":  ["Total Assets", "Total Liabilities", "Total
Equity"...],
  "Income Statement":  ["Operating Revenue", "Operating Costs", "Taxes
and Surcharges"...],
  "Cash Flow Statement":  ["Operating Activities", "Investment
Activities", "Financing Activities"...]  }
```

2.3 Time and Subject Labeling

This stage employs LLMs to label temporal information and subject classifications within the extracted data. The prompt instructs the model to:

```
 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 Extraction

The formula extraction phase uses a targeted prompt to identify all unique time points and subjects required for calculating the financial matrices. The prompt instructs the model to:

```
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

The value extraction phase uses a targeted prompt to identify specific financial metrics:

```
 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

For calculation reliability, I implemented a comprehensive error coding system. The calculation prompt includes specific instructions for error handling:

```
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 study utilizes a comprehensive dataset of Chinese listed companies' annual reports published after October 2023. The sample includes 5,442 companies, representing over 99% of all Chinese A-share listed companies. The source financial reports are in PDF format, containing three main financial statements: balance sheets, income statements, and cash flow statements. The ground truth for matrix values is obtained from CSMAR, a widely recognized database for Chinese financial market research.

Table 1: 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

3.2 Selected Matrices and Rationales

I selected 18 operating-related matrices categorized into five groups based on their business significance:

Table 2: 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

3.3 Overall Evaluation Results

The evaluation across different LLM models reveals varying levels of performance. The table below summarizes the results for exact match accuracy and performance within tolerance bands of ±1%, ±5%, and ±10%:

Table 3: Performance Evaluation of LLM Models

Model	**Exact Match**	**Within ±1%**	**Within ±5%**	**Within ±10%**
Sonnet	31.25%	50.00%	57.50%	71.25%
O1 Preview	20.84%	29.16%	31.95%	33.33%
Haiku	15.00%	15.00%	25.00%	40.00%
GPT-4	15.00%	15.00%	20.00%	30.00%
GPT-4 Mini	6.25%	6.25%	10.00%	16.25%
O1 Mini	0.93%	0.93%	3.70%	12.96%

3.4 Universal Strengths and Weaknesses

The evaluation reveals consistent patterns across all models when calculating financial matrices. Table 4 ranks the average performance of all matrices, while Figure 1 provides a detailed overview of model performance across different tolerance bands.

Table 4: Average Rank of Financial Matrices Across Models

Matrix	**Average Rank**
Accounts Receivable Revenue Ratio	2.8
Current Assets Revenue Ratio	5.3
Inventory Revenue Ratio	5.5
Fixed Assets Revenue Ratio	6.3
Accounts Receivable Turnover	6.8
Current Assets Turnover	8.0
Operating Cycle	8.8
Total Assets Turnover	8.8
Accounts Receivable Days	9.0
Capital Intensity	10.0
Fixed Assets Turnover	10.5
Inventory Turnover	10.8
Cash Turnover	11.8
Working Capital Turnover	12.8
Equity Turnover	13.0
Noncurrent Assets Turnover	13.3
Accounts Payable Turnover	13.5
Inventory Days	13.7

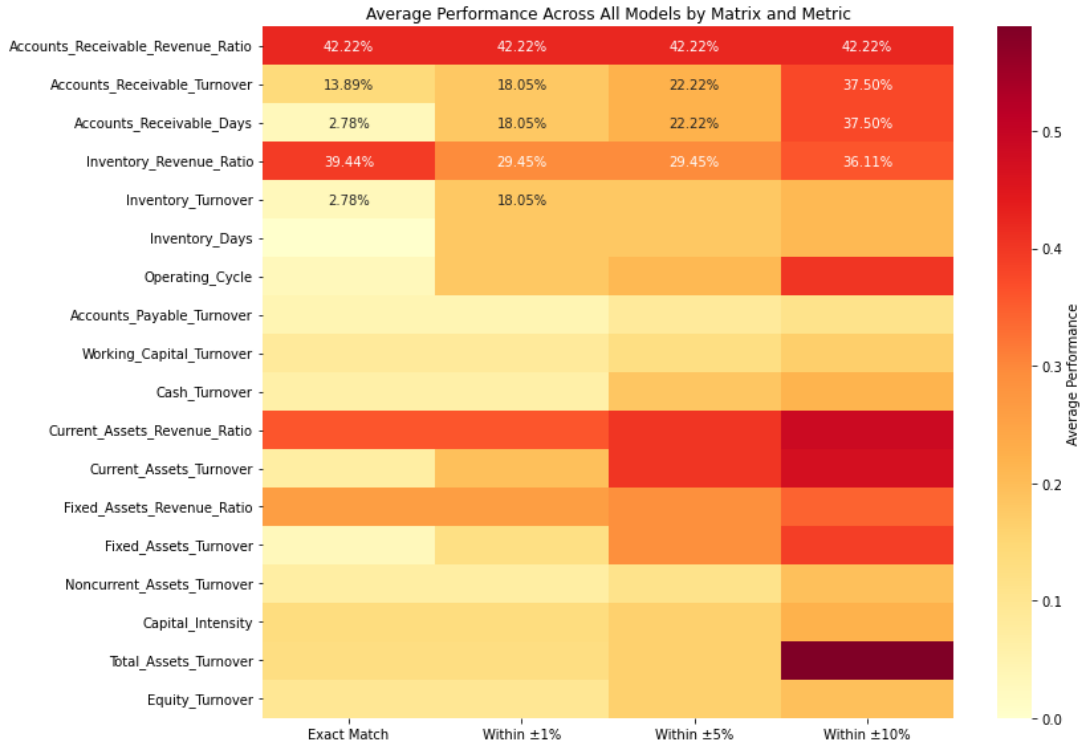

Figure 1: Average Performance Across All Models by Matrix and Metric

The evaluation reveals consistent strengths and weaknesses across all models when calculating financial matrices. **Models exhibit strong performance** on simple metrics that involve single-period data and direct one-step calculations. For instance, metrics such as *Accounts Receivable Revenue Ratio*, *Current Assets Revenue Ratio*, and *Inventory Revenue Ratio* consistently achieve high accuracy. These metrics have clear definitions and straightforward relationships, making them less prone to errors during computation.

Conversely, **models demonstrate consistent weaknesses** on complex metrics involving multi-step calculations, cross-period reasoning, or temporal alignment. Metrics like *Inventory Days*, *Accounts Payable Turnover*, and *Noncurrent Assets Turnover* rank the lowest. These matrices often require the averaging of values or interpreting relationships across different time periods, which increases both the complexity and likelihood of errors.

The observed strengths can be attributed to the simplicity of single-period computations, where numerical relationships are clear, and no additional reasoning steps are required. Such tasks are inherently easier for models to solve because they rely solely on extracting and computing data within a single table. In contrast, **the weaknesses arise** due to two primary factors: **cross-period reasoning** and **multi-step calculations**. Metrics like *Inventory Days* require calculations that span multiple time periods, which challenges the models' ability to align historical or future data. Similarly, metrics such as *Accounts Payable Turnover* involve sequential operations like division followed by averaging, which increase the likelihood of intermediate errors.

As illustrated in the heatmap (Figure 1), these challenges significantly impact model performance, particularly for metrics with higher computational complexity.

4. Conclusion

Through the development and implementation of the OFIN benchmark, I have demonstrated both the potential and current limitations of LLMs in financial matrix calculation. The best-performing model achieved 31.25% exact match accuracy, with performance improving to 71.25% when allowing for a $\pm 10\%$ tolerance. This indicates that while LLMs show promise in financial analysis, their capabilities are currently limited.

The study reveals several significant limitations in current LLM applications to financial analysis. Technical constraints include context window length limitations that restrict the amount of financial data that can be processed at once, coupled with restrictions on multi-modal input that affect the ability to process complex table structures. Additionally, limited input tokens per minute and response latency impact processing efficiency and real-time analysis capabilities. In terms of calculation accuracy, mathematics hallucination occurs in complex calculations, while models struggle with maintaining temporal context in cross-period calculations. Performance consistently degrades as calculation complexity increases.

For future research, I propose expanding the evaluation to the full dataset of 5,442 companies across all available models to provide more comprehensive benchmarking results. This expanded analysis should focus on improving the technical pipeline to better handle temporal context and complex calculations, while investigating methods to reduce token usage and latency without sacrificing accuracy. Furthermore, exploring the integration of domain-specific knowledge could enhance calculation accuracy.

The OFIN benchmark serves as a foundation for understanding and improving LLM capabilities in financial analysis, with potential applications extending beyond matrix calculation to broader financial analysis tasks.

Data Availability

The finanical matrices data from operational matrices library of CSMAR is available at the following site: https://data.csmar.com/.

Acknowledgments

Dear thanks to Professor Tianyi Peng and Hannah Li from Columbia Business School for holding the course Generative AI: Technical and Social, and their sincere help and

guidance during this project.

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. (2024). Financial statement analysis with large language models. *arXiv preprint*, 2407(17866).

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).

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).

listings xcolor

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.

```
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

```

```python
# 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)

# 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.

```
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.

```
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.

```
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"
        ]
    }

    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.

```
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
```

```python

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