2025

MUJTABA ASIF

TOMORO.AI

Table of Contents

Problem Statement

This project implements a multi-turn question answering system for financial reasoning tasks using the ConvFinQA dataset.

Models

- Qwen3/Qwen3-8b
- Meta_llama/LLaMA-3.1-8B

Challenges

In this section we review some of the most challenging aspects of this task. As they will be essential to come up with the final system design.

Multi-hop Reasoning Across Turns

Questions in a conversation often depend on previous questions and answers hence require chaining multiple reasoning steps to form a better context. For e.g. in this case, it can pose the following challenges:

- Requires memory of previous conversation (e.g., “What’s the growth compared to last year’s revenue?”).
- If an initial response is wrong, all dependent turns are most likely to fail.
- Standard LLMs are not designed to retain or reason over evolving multi-turn chains unless the information is provided as an external context.

Retrieval from Semi-Structured Tables

It is very difficult to identify and extract the correct values from financial tables. For e.g it can pose the following difficulties and complications:

- Tables have varied structures, missing headers, duplicate column names, and inconsistent formats (e.g., $1,234, (1234), -1234).
- Requires understanding both **row context** (e.g., "Operating Income") and **column context** (e.g., "2020 vs 2021").
- Errors in retrieval can lead into incorrect final answers.

Context Tracking in Multi-turn Dialogues

Knowing when to use prior answers, when to discard outdated context, and how to maintain reasoning continuity is essential to improve the overall quality of the multi-turn QA system. For e.g. it poses these challenges:

- User questions may refer implicitly ("What about the next year?").
- LLMs struggle to find the perfect balance between using too much and too little history.
- Invalid or incorrect context often leads to incorrect reasoning and answers.

Numerical Reasoning and Symbolic Operations

Questions often require arithmetic operations (e.g., additions, differences, percentages, exponentials) to compute the final answers.

- LLMs are prone to **hallucinating math steps** or making subtle calculation errors.
- Numeric precision matters in finance — even small mistakes can lead to large errors.
- The order of operations is very pivotal to extracting correct answers.

Solutions

In this section we will discuss the methods used to retrieve and compute the final answer. The section further contains the following parts: context building, prompt strategies, history, architecture.

Context Building

What: Constructing the full document context that includes table data pre_text and post_text.

How:

- The table is converted to a Markdown format enclosed within [Table Begin] and [Table End].
- It is combined with pre_text and post_text (text surrounding the table) to form a complete document view.

Why:

- ConvFinQA requires understanding both structured (tables) and unstructured (text) content.
- Accurate extraction depends on clear table cells and consistent structure.
- Ensures the LLM receives the complete information required for both retrieval and generation.

Prompting Strategies

What: Designing specialized system and user prompts for each step needed to answer a multi-turn financial question. A clear and well-defined prompt helps ensure accurate answers. It is essential to define responsibilities carefully in prompts, add guardrails, add reasoning steps to receive the best performance from an LLM.

How:

- **System Prompt**:
  Defines the assistant's role as a financial reasoning agent, explains the structure of the input document (tables + text), and outlines the expected step-by-step reasoning procedure.
- **User Prompts** vary based on the task:
  - **Question Reformulation Prompt**
    Prompt for rewriting current question into clear, standalone questions using prior conversion history.
  - **Retrieval Prompt**
    Prompt for extracting only the relevant rows, columns, or values from the table based on the reformulated question. It also tries to ensure that the prompt is only responsible for retrieving the relevant information from the document without performing any computations.
  - **Step Generation Prompt**
    Produces step-based operations when the answer requires computation (e.g., subtract (1234, 1000)). Prompt tries to ensure only predefined operators like add, divide, exp, multiply, subtract are used and while maintaining the order of operations.
  - **Formatting Prompt**
    Converts final answers into the appropriate format, such as a plain number, a percentage string, or a formatted date.
  - **Few-shot + Chain-of-Thought Prompting**
    To improve model performance, perform structured operations, reduce hallucination and ensure roles of each stage:
    - We provide **few-shot examples** for both **retrieval** and **program generation**, showing how to formulate the outputs with respect the provided context.
    - In step generation, the model is prompted to **think step-by-step** using **Chain of Thought prompting** before producing the final program with the aim of enhancing reliability on complex, multi-hop questions.

Why:

- Different subtasks (retrieval vs generation vs formatting) require distinct prompting strategies to be effective.
- Separation of prompt roles reduces cross-task confusion and makes debugging easier.
- Few-shot prompting allows the model to generalize better from examples that are provided as a reference.
- Chain-of-Thought prompting improves reasoning transparency and step accuracy by encouraging intermediate steps and allowing the LLM to retrospect its answers.
- Modular prompting enables error isolation, allowing component-level evaluation and targeted improvements.

History Tracking

What: Maintaining conversational memory across question turns is important to provide context to the further questions in Multi-hop QA systems.

How:

- For every turn, the system logs:
  - Full conversation history (for reformulation).
  - Retrieval and generation history (for conditioning prompts).
- Reformulation uses past Q&A pairs to rewrite questions as standalone.
- Retrieval and program generation can condition on previous operations or answers to reflect on next steps.

Why:

- Many questions in ConvFinQA are implicitly dependent on previous answers.
- Without history, questions like “What about the next year?” become unanswerable.
- Allows consistent reasoning and context-aware decision making.

Architecture

We implemented and compared two architectural approaches for solving the ConvFinQA multi-turn QA task. Each approach is modular and composed of distinct components, but they differ in execution strategy and structure.

Approach 1: Chain-of-Thought-QA

Explanation

This approach builds a Chain-of-Thought (CoT) based pipeline for financial multi-turn QA using local LLMs. It combines static CoT examples, table context, and history tracking to simulate a financial assistant that reasons step-by-step over each turn.

Steps

1. **Convert** financial tables to Markdown format and combine with pre_text and post_text to build the document context.
2. **Parse** the turn_program for each question into step-by-step operations (e.g., add, divide) for CoT reasoning.
3. **Construct prompts**:
   - A static **system prompt** that outlines expected reasoning behavior along with the detailed explanation around information structure.

    - A **user prompt** that includes full context, a CoT example (from record 0), and the current question.
4. **Simulate dialogue** by tracking Q&A history across turns.
5. **Query local models** (Qwen3-8B or LLaMA-3.1-8B) using OpenAI-compatible endpoints.
6. **Save model responses** (llm_answers) for evaluation and analysis.

Why This Approach

- Easy to prototype and establish the baseline
- To enforce transparent, interpretable reasoning via chain-of-thought.
- CoT improves traceability of reasoning steps.
- Maintains conversation flow using multi-turn history tracking.

Limitations

- CoT example is static (always from the first record), reducing generalization.
- Markdown-based table parsing may not be reliable for all model types.
- No explicit **question reformulation** ambiguous questions remain unresolved.
- Cannot evaluate **intermediate step quality**, limiting error tracing and modular evaluation.

Approach 2: Graph-Based Architecture (LangGraph)

Explanation

This approach builds a **modular, reasoning-aware agent** for multi-turn question answering on the ConvFinQA dataset. It integrates large language models (LLMs) with structured program generation and execution through a **graph-based workflow** using **LangGraph**. The agent mimics a financial assistant capable of interpreting complex tables, reformulating questions through query reformulation, retrieving relevant data from the tables, reasoning step-by-step, and producing well-formatted final answers.

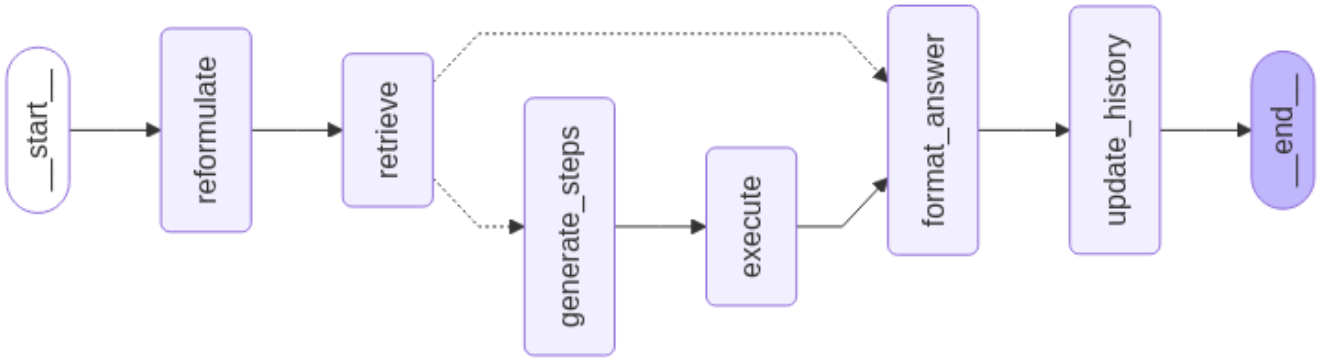

Figure 1

1. Question Reformulation

Rewrites ambiguous or context-dependent user questions to be fully self-contained using the conversation history. The step is pivotal because:

- Many questions in ConvFinQA are vague or incomplete ("What about next year?" or "How does that change?").
- Reformulation makes questions independent and explicit for better LLM understanding hence leading to better answers.

2. Retrieval

Finds and extracts only the relevant rows/columns or values from the context (i.e., the financial table and text). It is essential to constraint the role of this step by providing clear and concise instructions to only perform retrieval.

- Prevents the model from being distracted by irrelevant information.
- Keeps the working memory focused on what's needed for reasoning.

- Supports both direct-answer and multi-step reasoning workflows.

3. Conditional Branching (Early Exit)

If the retrieval result looks like a final answer (e.g., a single number), skip step generation and return the answer directly.

- Saves computation at the program generation and avoids performing unnecessary extra steps.
- If the answer is directly extracted from the provided context, no further steps are needed.

4. Step Generation

Uses a prompt to generate a program with operations like subtract, divide, multiply, add and exp, to compute the answer to user's query. It is essential for:

- Formalizing step-by-step program generation while maintaining the order of operations.
- Enabling interpretability and program execution for correctness checks.
- Breaks down complex questions into arithmetic/logical steps, supporting robust multi-hop reasoning.

5. Program Execution

Parses and runs the generated program to compute the final numerical result. Instead of relying upon LLMs we use internal method to perform computations. It helps with the following:

- Ensuring correctness and consistency in numeric outputs.
- Reducing hallucination risk by not relying on LLM to calculate values.

6. Answer Formatting

Formats the final output based on the type of question (number, percent, date, etc.). The is important to ensure the following:

- Align with evaluation metrics and user expectations.
- Ensures output consistency (e.g., "0.25" → "25%" if the question asks for a percentage). As incorrect formatting can be misunderstood as an incorrect answer.

7. History Update

Appends the current question and its answer to the ongoing conversation history. It helps with the following:

- Maintains conversational continuity for further questions.
- Supports coreference and follow-up resolution in future turns.

Advantages

- Can skip unnecessary steps based on retrieved content reducing computation on simple questions.
- Easier to inject fallback paths (e.g., retry program generation). Instead of recomputing all the steps.
- Each node is independently testable and explainable.
- Modular structure allows separation of concerns (retrieval vs generation vs formatting).
- Self-contained questions via question reformulation improve reliability of answers.
- Retrieval + generation separation reduces hallucination and improves transparency for each stage.

Possible Improvements

- Slightly more complex to set up and maintain.
- Requires more careful tracking of state transitions and variables.
- Needs robust confidence checks to avoid incorrect early exits.
- Sequential steps can lead to higher latency.

Evaluation Benchmarks

LLMs were used to evaluate the responses of the models. These results are computed on the dev split.

Summary

- Correct program generation is the strongest indicator of answer accuracy.
- Retrieval quality heavily influences both program generation and final answers.
- Skip logic is not performing well meaning it needs improvements.
- Performance drops across dialogue turns, showing lower accuracy due to increased context complexity.

Answer Correctness

- Exact Match (EM) tells us whether the final answer exactly matches the expected answer critical in financial QA where the exact response is essential.
- The exact match correctness accuracy is **60.1%.**

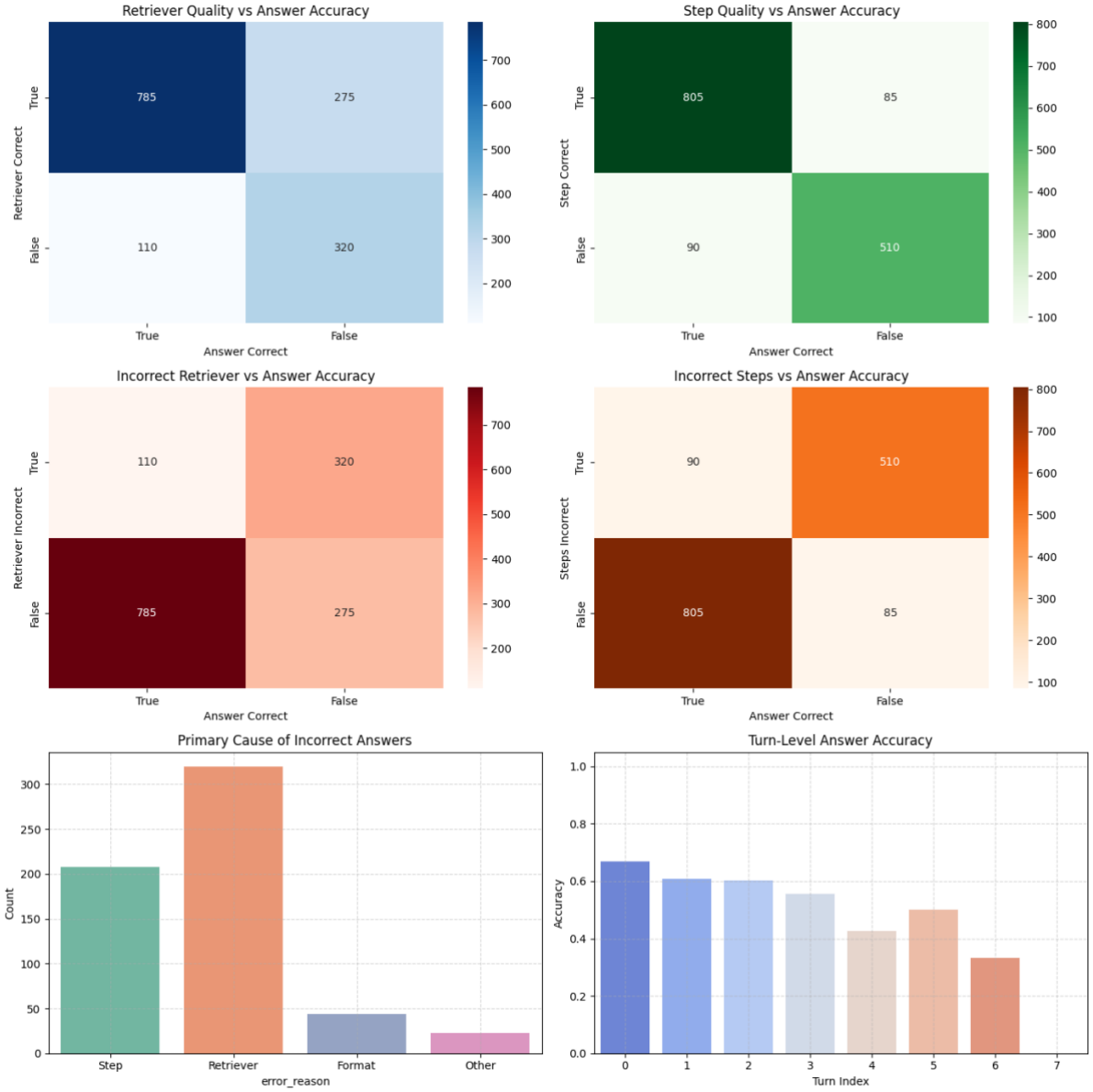

Figure 2

Findings:

The most important findings from Figure 2

- **Retriever Quality vs Answer Accuracy:**
  - Good retrieval results in more correct answers (785 correct vs 275 incorrect).
  - Retrieval failures strongly correlate with answer failures.
- **Program Quality vs Answer Accuracy**
  - Correct reasoning steps lead to high answer accuracy (805 correct vs 85 incorrect).
  - Incorrect steps often result in wrong answers (510 incorrect).
- **Primary Cause of Incorrect Answers**
  - Most errors are due to: 1) Retrieval issues, 2) Step generation errors, 3) Formatting problems (least frequent).
- **Turn-Level Answer Accuracy**
  - Accuracy drops with each turn: from ~65% on Turn 0 to ~35% on later turns.

Retrieval Scores

These metrics evaluate the retriever's ability to extract relevant information from the document context. They measure how retrieval correctness correlates with final answer accuracy, both in simple (no reasoning needed) and complex (reasoning required) cases.
The retriever accuracy is **71.1%,** precision **74.1%,** and recall is **87.7%**.

Figure 3

Findings

The most important findings from Figure 2

- **Retriever Quality with no code generation**
  - Retriever is correct in most direct-answer cases (~425 vs ~65) where the code generation is not required.
  - If retrieval is correct, and no code generation is skipped answer is always correct.
- **Retriever Quality with all samples**
  - 88% of correct answers had correct retrieval.
  - 54% of wrong answers were due to bad retrieval.

Retrieval quality is a strong indicator of overall Question Answering success.

Program Generation

Program generation scoring is important because it helps identify whether the generation logic produced by the LLM is valid and sufficient to deduce the correct answer. It reveals where failures occur due to bad retrieval, poor generation, or incorrect decisions to skip generation.
The program generation accuracy is **59.7%**

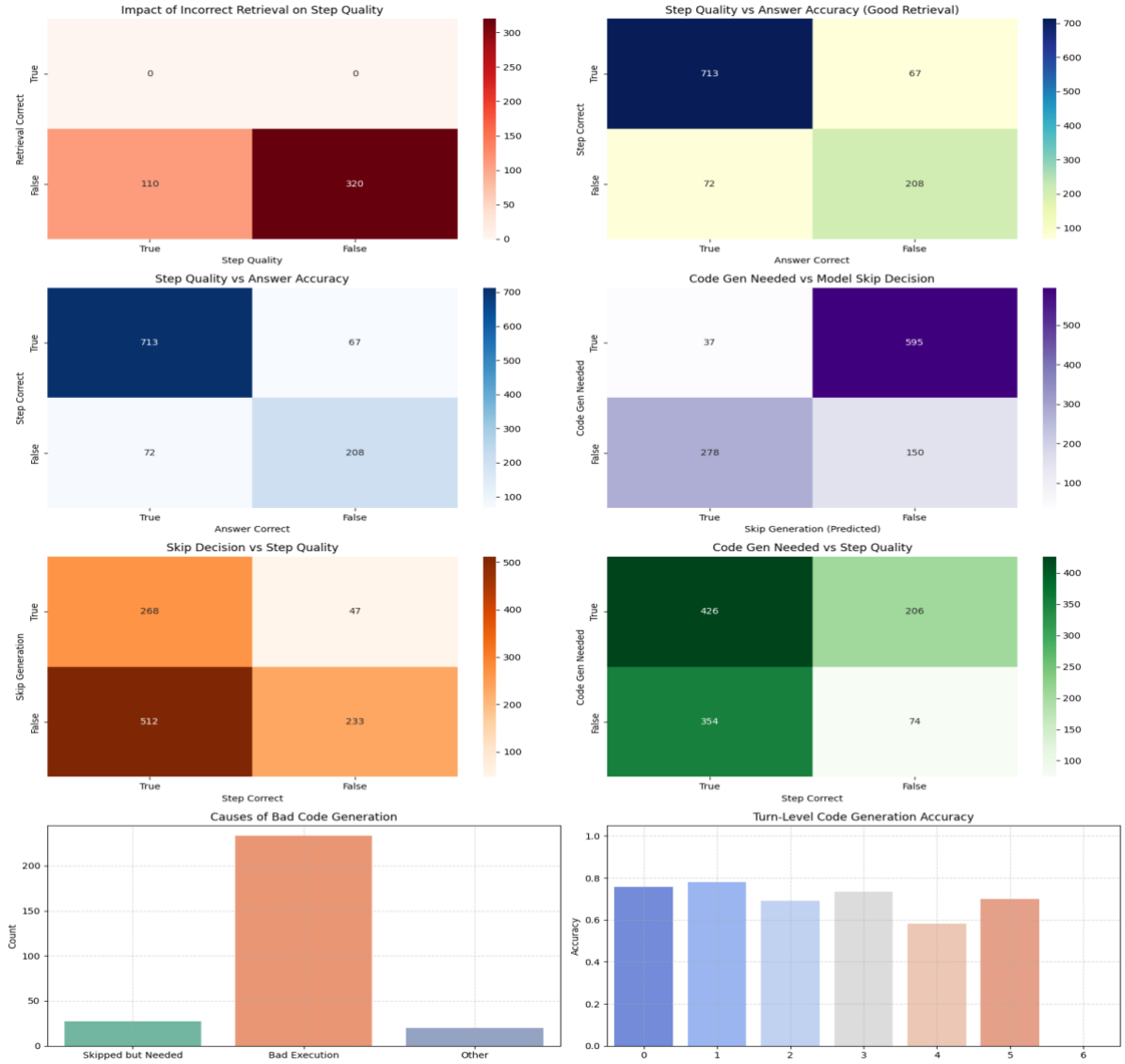

Figure 4

Findings

The most important findings from Figure 3

- Bad retrieval leads to high failure in step generation (75%).
- Correct reasoning steps are strongly correlated with correct answers (91.5%).
- Program generation skipping step has a very high error rate (will require a significant reimplementation).

Correlation Between Evaluation Metrics

Pearson correlation between four evaluation metrics:

- **llm_answer_correct**: Whether the final answer was correct
- **llm_retriever_quality**: Whether retrieval was correct
- **llm_step_quality**: Whether the reasoning steps were correct
- **llm_format_match**: Whether the final answer matched the expected format

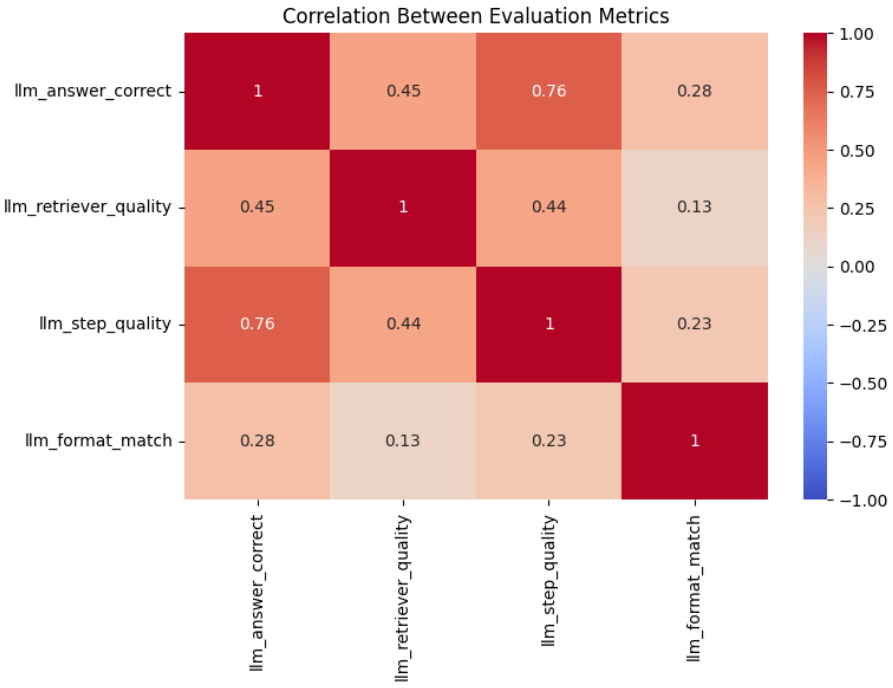

Figure 5

Findings

Refer to the Figure 5 for more detailed values

- **Answer and Step Quality:** They have the strongest correlation meaning correct program generation steps lead to most correct answers.
- **Answer and Retriever Quality:** They have a moderate correlation, and good retrieval is essential for correctness, but it also requires support from code generation.
- **Answer and Format Match:** They have a very weak correlation meaning formatting has minimal impact on answer correctness.
- **Step and Retriever Quality:** Retrieval quality influences program generation quality meaning bad retrieval often leads to incorrect program generation.
- **Format and All Others:** Format match has little to no impact on other components and behaves independently.

Conclusion

To improve the system, future work should prioritize:

- Enhancing the robustness of step generation, especially for cases where the retrieval is incorrect.
- Developing adaptive retrieval and code generation triggers based on context and question type.
- Integrating confidence-based fallback strategies for low-quality retrieval or reasoning outputs.

Overall, the current architecture provides a solid foundation, and targeted improvements to retrieval and reasoning modules can yield substantial gains in performance and reliability.

Further Improvements

- **Better LLM**
  - Since, the current solution relies on Qwen3 and Llama3.1 using a better model can significantly boost the overall accuracy.
- **Better Table Representation**
  - Replace Markdown with structured formats or use structure-aware models (e.g., TAPEX, TaPaS).
- **Few-Shot & Prompting Improvements**
  - Dynamically select similar CoT examples (e.g., via embedding similarity or clustering).
  - Improve prompts to support deeper, multi-step reasoning chains.
- **Model Finetuning**
  - Finetune with instruction-tuned data specific to financial reasoning.
- **Agentic Reasoning & Control Flow**
  - Use agentic workflows like ReAct for dynamic reasoning and tool use.
  - Extend the graph with verification, explanation, and feedback nodes.
- **Robust Error Handling**
  - Detect invalid programs and regenerate as needed.
  - Add recovery logic for skipped steps or failed execution.