Large language model number handling in the Finance Domain

Anant Raj

Master of Science
School of Informatics
University of Edinburgh
2024

Abstract

The integration of large language models (LLMs) in the financial domain requires scrupulous attention to numerical precision and the ability to process diverse data types. In this study, we enhance the capabilities of open-source Llama models through an innovative fine-tuning approach that leverages a hybrid dataset, combining tabular data, conversational histories, and contextual information. We employ specialized prompting, few-shot learning, and Chain of Thought reasoning to address complex numerical tasks inherent to financial analysis. Furthermore, we guide the models to generate Python scripts aimed at bolstering numerical accuracy. Our approach offers valuable insights into the potential and challenges of LLMs in complex numerical reasoning within financial contexts, illuminating areas for further enhancement. Our findings highlight the ongoing difficulties in merging varied data forms within LLM frameworks and underscore the necessity for continued refinement of these models to meet the rigorous demands of financial data analysis. This research not only advances our understanding of LLM applications in finance but also sets a foundational framework for future explorations aimed at enhancing the robustness and precision of financial data processing.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Anant Raj*)

Acknowledgements

Embarking on this thesis journey has been an exhilarating experience, and I am profoundly grateful to those whose support has been indispensable. My heartfelt thanks to Professor *Alexandra Birch*, whose unwavering commitment and insightful guidance have been cornerstones of my research. Her scholarly rigor and the access she facilitated to the Valhalla server and the statmt slack community were pivotal during critical phases of my work. Special thanks to *Mateusz*, whose innovative ideas significantly streamlined my research process. Additionally, I owe a tremendous debt of gratitude to my sister, *Priya*, whose boundless patience and encouragement, especially across challenging time zones, have been invaluable. Her willingness to listen and her unyielding presence have been my comfort and strength. This dissertation is not just a product of my individual effort but a testament to the collective support of everyone who cheered me on. Thank you all for being part of my journey.

Table of Contents

Chapter 1

Introduction

The intersection of Large Language Models (LLMs) and the financial industry represents a promising yet challenging frontier. LLMs such as GPT-4 and Claude AI have demonstrated significant capabilities across various Natural Language Processing (NLP) tasks, driven by advanced training methods like reinforcement learning from human feedback (RLHF) and masked language modeling [24]. However, these models also have notable limitations, including difficulties with current information access, arithmetic tasks, and a tendency to generate inaccurate responses due to hallucinations [34, 22]. Despite being trained on diverse datasets, their performance in specialized domains like finance remains underexplored. LLMs are increasingly integral to financial applications such as investment sentiment analysis, named entity recognition, question-answering, and stock market prediction. These tools assist analysts in managing complex datasets and predictive models. Methods like multi-field LLMs and instruction fine-tuning have been explored to improve results in financial contexts [18]. However, the nuanced understanding and processing of complex financial data by LLMs are still in early stages, necessitating further empirical study.

This research investigates various fine-tuning methodologies for open-source language models, focusing on instruction tuning, few-shot prompting, and chain-of-thought reasoning. Previous studies have demonstrated that these techniques can enhance performance across diverse tasks, including translation, question-answering, and complex reasoning [42, 17, 5]. Our study aims to apply these methods to finance-specific tasks that require numerical computation and complex reasoning. We utilize four diverse financial datasets, which include tabular data, contextual information, relation extraction, and conversational finance question-answering, to challenge the models and enhance generalizability across various financial tasks. The diversity of the datasets poses a

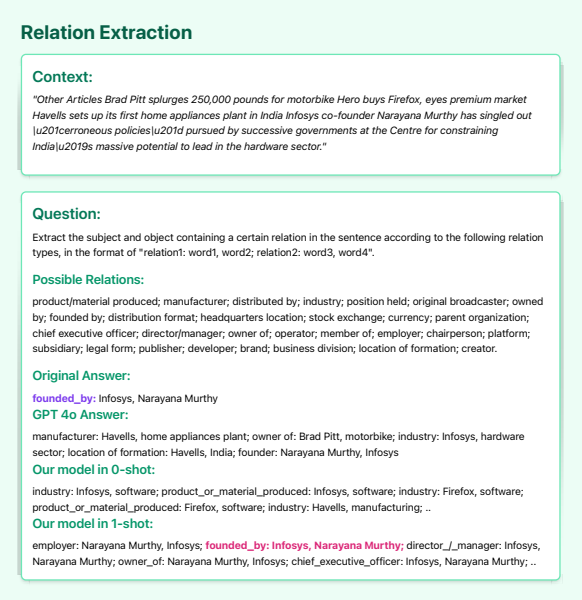

Figure 1.1: Relation extraction sample with GPT-4o and our model in 0-shot and 1-shot.

unique challenge, as there is no one-size-fits-all approach that can be applied effectively. For example, few-shot prompting (FSP) performs well on many tasks but may struggle with more complex reasoning tasks, which constitute three out of the four datasets in this study (TAT-QA, ConvFinQA, and FinQA). These datasets require a high level of reasoning, arithmetic solving capability, and precise extraction of tabular data. The relation extraction dataset further adds the generalization required to build a robust model for general usage. Figure 1.1 illustrates the performance of the latest GPT-4o model in relation extraction tasks. Despite its advancements, GPT-4o often generates multiple extractions per query, whereas the reference contains only one. Conversely, our model in one-shot, produces the correct extraction but also includes additional extractions. This issue stems from the dataset's ambiguity regarding the acceptable number of responses, allowing multiple relevant extractions as per the context and question. In relation extraction, the significance of input tokens varies. However, standard prompt templates for classification treat all words equally, failing to reflect this differential importance effectively [29]. Additionally, the challenge of instruction tuning arises, necessitating the creation of high-quality instructions for fine-tuning to address the complexities introduced by variations across different datasets. Even within a single dataset, such as TAT-QA, the level of difficulty varies, with some examples requiring only data extraction from tables (for sample A.2.1), while others necessitate additional calculations to generate the final answer. Similarly, for ConvFinQA, the challenges lie in understanding the textual and tabular data from the input and keeping track of

the question-answer history of the conversation to provide the correct response(see Figure 1.2). Furthermore, the variable length of the input, with ConvFinQA having particularly long inputs, requires the model to process the full context without truncation to generate accurate responses.

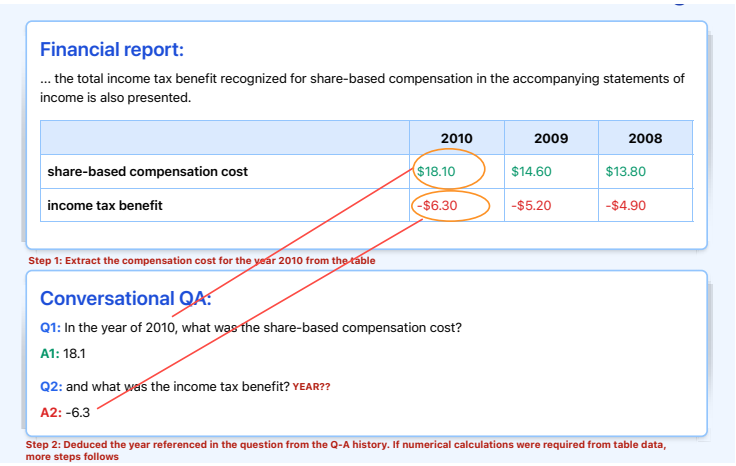

Figure 1.2: QA History in ConvFinQA

Initial experiments with open-source models like LLaMA2 and LLaMA3 confirmed that LLMs often struggle with numerical reasoning, relation extraction and hybrid tasks. Various fine-tuning approaches, including FSP, chain-of-thought (CoT), and instruction tuning, were explored. Ultimately, a combination of FSP and CoT proved most effective in addressing the dataset's challenges. This approach, particularly novel in the financial domain, builds on previous works [16, 52] but extends their scope to include conversational question-answering and relation extraction tasks, further enhancing the model's robustness and generalizability.

In our next study, we extend the innovative principles of Schick et al. (2023) from their seminal work *Toolformer: Language Models Can Teach Themselves to Use Tools*, which demonstrated the enhancement of language models' performance by integrating external tools like calculators and search engines [34]. Inspired by this, our study introduces a sophisticated pipeline that integrates LangChain agents and function calling to address the challenges posed by complex datasets like TAT-QA. This approach employs a combination of few-shot prompting (FSP), instruction tuning, and Chain of Thought (CoT) reasoning. By leveraging these strategies, our methodology enables language models to independently generate and execute Python code, thus significantly enhancing their reasoning and computational capabilities for complex financial tasks.

The pilot study utilized the FACEBOOK/OPT-1.3B model in conjunction with LangChain's Python agent to produce promising results. This was further scaled to the more powerful Llama-3.1-8b-instruct model, handling more complex datasets like TAT-QA. These experiments underscore the pivotal role of model size and computational power in processing demanding financial datasets effectively.

Our research not only advances the capabilities of language models in financial applications but also discusses critical aspects of scalability and the balance between computational power and model sophistication. Our findings suggest that with strategic implementation of code generation pipeline, LLMs can achieve state-of-the-art performance in financial tasks, paving the way for future advancements in the use of open-source models for financial analysis.

Contributions: Our research introduces a series of novel methodologies and results that significantly advance the application of language models in financial data analysis. First, we uniquely combine few-shot prompting (FSP) and Chain of Thought (CoT) reasoning for finance-specific tasks, which has demonstrably improved model performance. This integration is specifically tailored to address the nuanced requirements of financial datasets, facilitating a deeper and more precise analysis. Additionally, we have curated a selection of four comprehensive financial datasets, a novel approach in this field that establishes a robust platform for generalizing language model capabilities across diverse financial applications. This dataset selection is instrumental in assessing the adaptability and scalability of our proposed methodologies. Further enhancing our research contribution, we introduced a novel evaluation metric, termed *Permissive Accuracy*. This metric is designed to accommodate minor discrepancies in precision, defined by a tolerance parameter α, or allows for a degree of leniency in string comparisons, defined by β . This metric is particularly useful in scenarios where exact precision is less critical than the overall trend or pattern recognition. Our modifications to the LLaMA3 8B model using the FSP and CoT approach have led to significant performance enhancements, with average permissive accuracy improving by 87.88%, average exact accuracy by 220.00%, ROUGE score by 90.00%, BLEU score by 133.33%, METEOR Score by 63.16% and GLEU score by 100.00%. These improvements underline the efficacy of our methodologies in enhancing the model's understanding and processing of complex financial information. Lastly, by employing LangChain's agent alongside Python functions, we have developed a pioneering pipeline for code generation and execution. This innovation significantly augments the model's computational and reasoning efficiency, showcased through our work on complex financial datasets like TAT-QA.

Chapter 2

Background

The field of natural language processing (NLP) has experienced a profound transformation in recent years, primarily propelled by the introduction of transformer-based models and the subsequent development of large language models (LLMs). This chapter presents an overview of the evolution of these models. Section 2.1 introduces the architecture of transformers, detailing their fundamental components and operational principles. Section 2.2 delineates the progression of LLMs, outlining key milestones and innovations and also examines the recent advancements in LLMs capabilities, showcasing the growing efficiency and versatility of these models in handling complex tasks across diverse domains.

2.1 The Transformer Architecture: A Paradigm Shift

The introduction of the Transformer architecture by Vaswani et al. (2017) represented a significant shift in the field of natural language processing (NLP), fundamentally altering how sequence transduction problems are approached. Unlike the earlier recurrent or convolutional neural network (CNN) architectures, the Transformer is built entirely around attention mechanisms, which allow for more efficient processing of sequential data and facilitate the training of substantially larger models [38]. The core components of the Transformer architecture include the self-attention mechanism, which assesses the relevance of different parts of the input sequence for each element, thus effectively capturing long-range dependencies. Additionally, the architecture employs multi-head attention to concurrently process various relationships within the data. Positional encoding is integrated into input embeddings to compensate for the architecture's lack of inherent sequential processing, thereby preserving the order of the sequence. Feed-

forward networks enhance the model by processing the outputs from the attention layers, introducing non-linearity and augmenting the model's ability to learn complex functions. Lastly, layer normalization and residual connections are crucial in stabilizing the learning process and preventing the vanishing gradient problem, thereby supporting the training of deeper networks. These innovative features collectively enhance the model's performance, enabling it to handle complex language processing tasks more effectively. The Transformer's ability to process input sequences in parallel, rather than sequentially, led to significant improvements in training efficiency and model performance. This architecture laid the groundwork for the development of increasingly large and sophisticated language models.

2.2 Evolution and advancement of Large Language Models

Building upon the Transformer architecture, researchers developed a series of increasingly powerful language models. BERT (Bidirectional Encoder Representations from Transformers), introduced by Devlin et al. (2019)[9], represented a significant advancement in pre-training techniques. BERT's bidirectional context consideration and novel pre-training tasks led to state-of-the-art performance across a wide range of NLP tasks. The GPT (Generative Pre-trained Transformer) series, developed by OpenAI, further pushed the boundaries of model size and capabilities. GPT-3, with 175 billion parameters, marked a leap in scale and showcased strong few-shot learning abilities, reducing the need for task-specific fine-tuning [5]. As LLMs evolved, several key advancements emerged in fine-tuning techniques and task-specific adaptations. Few-shot learning, demonstrated by Brown et al. (2020) with GPT-3, showed that sufficiently large models could perform tasks with minimal task-specific training examples [5]. This capability opened new possibilities for creating more flexible and adaptable AI systems and we have utilised this approach in our methodology to fine tune the financial dataset as detailed in section 4.3. Instruction tuning, introduced by Wei et al. (2022), showed that fine-tuning models on diverse sets of instructions could significantly improve their ability to follow natural language prompts, enhancing the versatility of LLMs across various tasks [42]. Chain-of-thought (CoT) prompting, another significant advancement proposed by Wei et al. (2022), demonstrated that prompting models to generate step-by-step reasoning could substantially improve their performance on com-

plex tasks, including those involving numerical reasoning[43]. This technique showed that LLMs could be guided to break down complex problems into more manageable steps, mirroring human problem-solving processes. This idea has been utilised in our code generation approach detailed in section 4.5.

The field of Natural Language Processing (NLP) has witnessed a remarkable evolution, driven by innovative fine-tuning methodologies that enhance model adaptability across diverse domains while utilizing fewer parameters. These methods not only preserve the performance standards of fully trained models but often surpass them, demonstrating the potential to streamline and optimize NLP applications. One notable innovation is prompt tuning, pioneered by Lester et al. (2021), which refines the approach to task-specific model optimization. By introducing continuous prompts that guide the pre-trained model—whose parameters remain unchanged—this technique offers a more parameter-efficient alternative to traditional fine-tuning [19]. Complementing this, In-context learning (ICL), as discussed by Liu et al. (2021, 2022), embeds task-specific examples directly within the input prompt, enabling models to adapt without altering underlying parameters. However, this method requires significant computational resources due to the necessity of processing extensive training data for each prediction [29, 28]. The versatility of large language models (LLMs) is further exemplified through their application in specialized fields such as healthcare and education. In healthcare, Singhal et al. (2022) have successfully employed LLMs for tasks like medical question answering and clinical decision support, showcasing their capability to handle sensitive and complex queries [36]. In the educational sector, Zhao et al. (2021) highlight the potential of LLMs in personalized tutoring and automated essay grading, suggesting a transformative impact on educational methodologies [50]. Recent technological advancements aim to refine the factual accuracy and reasoning capabilities of LLMs, pivotal aspects of our research. Innovations such as the self-consistency decoding strategy by Wang et al. (2023), which enhances performance by selecting the most consistent answers from multiple reasoning paths, and the decomposed prompting approach by Khot et al. (2023), which simplifies complex reasoning tasks into smaller, manageable components, are instrumental in advancing our understanding of effective model application [40, 15]. In summary, the landscape of NLP is rapidly advancing, characterized by significant breakthroughs that promise to revolutionize the field through efficiency and versatility. This surge in technological innovation lays a robust foundation for our study, enabling us to explore and potentially harness these advancements to further our research objectives.

Chapter 3

Related Work

The application of Large Language Models (LLMs) in the financial sector has been a subject of intense research in recent years, with studies exploring their potential and limitations across various financial tasks. This chapter reviews key works in this field, highlighting methodologies, findings, and persistent research gaps. General-purpose LLMs like GPT-3, ChatGPT, and GPT-4 have demonstrated broad capabilities across various financial tasks. Li et al. (2023) conducted a comprehensive evaluation of these models using multiple benchmark datasets [25]. Their methodology involved fine-tuning these models on financial data and comparing their performance against specialized financial models. While the general-purpose LLMs often outperformed specialized models in tasks like sentiment analysis and named entity recognition, they struggled with complex financial reasoning tasks. This highlights a significant research gap: the need for LLMs that can perform deep, domain-specific financial analysis while maintaining their general language understanding capabilities. Recognizing these limitations, researchers have begun developing models specifically tailored for the financial domain. Wu et al. (2023) introduced BloombergGPT, a 50-billion parameter language model trained on a vast corpus of financial data [44]. Their approach involved pretraining the model on a mix of general and financial text data, followed by fine-tuning on specific financial tasks. BloombergGPT demonstrated significant improvements over general-purpose models in tasks such as financial sentiment analysis and question answering. However, the authors noted that the model still struggled with complex numerical reasoning, highlighting a persistent gap in LLMs' ability to perform accurate financial calculations consistently. Wang et al. (2023) proposed FinGPT, leveraging instructional tuning to adapt LLMs for financial applications [39]. Their methodology involved fine-tuning LLMs on a diverse set of financial instructions and tasks. The

authors emphasized the importance of systematic evaluation across various financial competencies, from basic tasks to complex multitasking scenarios. While FinGPT showed improvements on text-based financial data, it overlooked other critical financial data forms such as numerical and tabular data, which are pivotal for comprehensive financial analysis.

Efforts to enhance LLM capabilities for finance have taken various forms. Chen et al. (2022) introduced the ConvFinQA dataset, focusing on complex numerical reasoning in conversational finance [7]. Their work involved creating a dataset of multi-turn financial conversations that require chained reasoning over numbers. They experimented with both neural-symbolic approaches and prompting-based methods, finding that even state-of-the-art LLMs struggled with complex, multi-step financial calculations. This work highlights a critical research gap: the need for LLMs that can perform reliable, multi-step numerical reasoning in financial contexts. Addressing the challenge of improving LLMs' instruction-following capabilities, Wang et al. (2023) proposed Self-Instruct, a method for enhancing LLMs through self-generated instructions [41]. While not specifically focused on finance, their approach of using LLMs to generate their own training data could potentially be applied to financial domains. The authors demonstrated a 33 percent improvement in GPT-3's general instruction-following capabilities. However, they noted limitations in the quality and diversity of self-generated instructions, pointing to a research gap in developing more sophisticated self-improvement mechanisms for LLMs. Araci (2023) introduced FinBERT, a BERT-based model specifically designed for financial sentiment analysis [2]. The author's methodology involved pretraining BERT on a large corpus of financial texts and fine-tuning it on labeled financial sentiment data. While FinBERT showed improvements over general-purpose models in sentiment classification tasks, it was limited to sentiment analysis and did not address more complex financial reasoning tasks, highlighting the need for more versatile financial LLMs. The challenge of context modeling and reasoning in LLMs, crucial for many financial applications, was addressed by Zhao et al. (2023) in their work on natural language-based context modeling and reasoning with LLMs [46]. They outlined various strategies for improving LLMs' ability to understand and utilize context, including prompt engineering, few-shot learning, and retrieval-augmented generation. While their work was not finance-specific, their findings have significant implications for financial applications. For instance, improved context modeling could enhance LLMs' ability to understand complex financial narratives or multi-turn financial conversations. Addressing the computational challenges associated with fine-tuning

large models for specialized applications, Dettmers et al. (2023) introduced QLoRA (Quantized Low-Rank Adaptation), a parameter-efficient fine-tuning methodology [8]. Their technique involves quantizing the pre-trained language model to 4 bits, integrating trainable low-rank adapters, and employing a novel preconditioner referred to as ”Paged Optimizers” to navigate GPU memory limitations. This method offers a feasible strategy for efficiently adapting large, general-purpose language models to specific financial tasks, significantly reducing the necessity for extensive computational resources. We adopted this approach in our research to fine-tune language models effectively within a resource-constrained setting.

Recent advancements in LLM capabilities have opened new avenues for their application in finance. Schick et al. (2023) introduced the Toolformer approach, which enables language models to learn to use external tools through self-supervised learning [34]. Their methodology involved augmenting a pretraining dataset with tool-use examples, allowing the model to learn when and how to call external tools. While not specifically focused on finance, this approach has significant implications for financial applications. For instance, a Toolformer-like model could potentially learn to access real-time market data, financial calculators, or regulatory databases, addressing the research gap of integrating external financial tools and data sources with LLMs. However, the authors noted challenges in tool selection and result interpretation, highlighting a persistent gap in LLMs' ability to reason about tool outputs which could be more pronounced in a complex domains like finance.

Chapter 4

Methodology

This chapter outlines the methodology used in this research project. Section 4.1 details the dataset selection and preparation process. Section 4.2 describes the computational models employed. Section 4.3 explains the fine-tuning techniques applied to optimize model performance. Section 4.4 discusses the evaluation metrics and iterative refinement processes. Finally, Section 4.5 covers the integration of external tools and code generation techniques, essential for automating and scaling our models' applications.

4.1 Dataset Selection and Preparation

4.1.1 Merged Dataset Creation

We have used a mixture of four different financial datasets. The brief summary of these dataset is provided below. The structured sample for each of these dataset after preprocessing is provided in A.2.

TAT-QA contains 16,552 questions associated with 2,757 hybrid contexts from real-world financial reports. The questions typically require a range of data extraction and numerical reasoning skills, including multiplication, comparison, sorting, and their various combinations [51].

ConvFinQA contains 3,892 conversations consisting 14,115 questions from real-world scenario of conversational question answering over financial reports. The dataset is formulated using both textual content and structured table [7].

FinQA is an expert annotated dataset that contains 8,281 financial QA pairs, along with their numerical reasoning processes. The reasoning processes answering these questions are made of many common calculations in financial analysis, such as addition,

comparison, and table aggregation [6].

FinGPT FinRED-RE dataset available on Hugging Face is designed for financial relationship extraction tasks. It contains 13.5k rows of data, divided into 11.4k training rows and 2.14k test rows. The dataset features text inputs with associated financial entities and their relationships. The task contains instructions for extracting financial relationships from textual data, utilizing specific relations like employer, industry, and product/material produced [1].

Merged dataset: The datasets are merged to form a comprehensive unified dataset, balancing various types of financial data to ensure robust coverage of multiple scenarios and data formats encountered in the financial domain. This integration includes tabular data, text-based financial reports, conversational question-answer pairs, and extracted relationships, enhancing the dataset's versatility and applicability for a wide range of financial data analysis tasks. This comprehensive approach improves the ability to analyze diverse financial situations and generalize across tasks, making it valuable for both financial domain applications and general numerical reasoning scenarios.

4.1.2 Dataset Preprocessing

In the development phase, a preprocessing algorithm (as referenced in Algorithm 1) was formulated to systematically organize the consolidated dataset, thus preparing it for subsequent model training. This algorithm is critical in ensuring that the data adheres to a standardized format, which includes distinct sections for context, questions, and answers, facilitating uniform data handling across various analytical procedures. Furthermore, the dataset was tailored to meet the specific requirements of the Llama2-7b and Llama3-8b models employed in this study. This customization involved several key processes: removing data inconsistencies and extraneous information to enhance data quality, standardizing numerical and textual elements to ensure consistency, and restructuring the data into model-specific prompts. This restructuring not only aligns with the models' training frameworks but also incorporates special tokens and prompt structures crucial for effective model fine-tuning, as detailed in the official documentation [2].

[1] https://huggingface.co/datasets/FinGPT/fingpt-finred-re

[2] https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3

Algorithm 1 Preprocessing Merged Dataset

procedure PREPROCESSDATASETS
    Load TAT-QA, ConvFinQA, FinQA, and RelEx datasets
    $combined\_data \leftarrow \emptyset$
    for $dataset \in \{TAT\text{-}QA, ConvFinQA, FinQA, RelEx\}$ **do**
        $processed \leftarrow$ PROCESSDATASET($dataset$)
        $combined\_data \leftarrow combined\_data \cup processed$
    end for
    $combined\_data \leftarrow$ RANDOMSAMPLE($combined\_data, \lfloor |combined\_data|/3 \rfloor$)
    Split $combined\_data$ into $train\_data$ (90%) and $test\_data$ (10%)
    return $train\_data, test\_data$
end procedure
procedure PROCESSDATASET($dataset$)
    $processed \leftarrow \emptyset$
    for each $sample \in dataset$ **do**
        $(context, question, answer) \leftarrow$ EXTRACTINFO($dataset, sample$)
        $processed \leftarrow processed \cup \{(context, question, answer)\}$
    end for
    return $processed$
end procedure
procedure EXTRACTINFO($dataset, sample$)
    if $dataset = TAT\text{-}QA$ **then**
        $context \leftarrow$ Combine paragraphs and format table
        for each $q \in sample.questions$ **do**
            return $(context, q.question, q.answer)$
        end for
    else if $dataset = ConvFinQA$ **then**
        return $(sample.input, sample.instruction +$ instructions$, sample.output)$
    else if $dataset = FinQA$ **then**
        $context \leftarrow$ Combine pre_text, post_text, and table
        return $(context, sample.qa.question, sample.qa.answer)$
    else if $dataset = RelEx$ **then**
        return $(sample.input, sample.instruction, sample.output)$
    end if
end procedure

4.2 Baseline Model Selection and Evaluation

The research utilized two baseline models, namely meta-llama/Llama-2-7B-chat-hf[3] and meta-llama/Meta-Llama-3-8B[4], to assess the initial performance metrics of the merged dataset. This performance testing phase was instrumental in establishing a benchmark for future comparisons, particularly to evaluate the enhancements post fine-tuning. Additionally, it provided an opportunity to discern the inherent strengths and limitations of the models when applied to unprocessed financial data. This baseline assessment is crucial for identifying potential areas of improvement and ensuring that subsequent adjustments are targeted and effective.

4.3 Fine-tuning Approaches

4.3.1 Training Procedure

Model Architecture

We employed the Meta-Llama-2-7B-chat-hf and Meta-Llama-3-8B model, a variant of the LLaMA architecture specifically adapted for few-shot learning scenarios. This choice was driven by its recent success in various NLP tasks, particularly those requiring nuanced understanding and generation based on limited context. For the code generation and execution task, we have also used the recently launched Meta-Llama-3.1-8B model due to its enhanced ability for code generation tasks.

Dataset Preparation

For the first experiment of fine tuning, we curated a dataset from multiple sources as mentioned in section 4.1, aiming to encompass a wide range of financial topics to enhance the model's ability to generalize across diverse financial contexts. The data was split into 90% for training and 10% for validation, ensuring a representative distribution of topics in each subset.

For the Code Generation experiment, we exclusively used the TAT-QA dataset, following a preprocessing strategy similar to the one outlined in the algorithm 1, specifically from steps 20-25.

[3]https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
[4]https://huggingface.co/meta-llama/Meta-Llama-3-8B

Training Details

The model was fine-tuned using a quantization-aware training approach, utilizing 4-bit quantization to balance performance with computational efficiency. The BitsAndBytes library facilitated the integration of low-precision arithmetic during training. Quantization is a two-step process, involves normalizing constants to scale vectors into a target range and rounding to the nearest target value. This technique can cause significant quantization loss if weights have outliers. To address this, *bitsandbytes* employs vector-wise quantization and mixed precision decomposition, maintaining performance akin to non-quantized states [12]. Although LLM int8 does not impair performance, it increases inference time due to quantization overhead but significantly reduces memory usage by 71%, which enabled us to fine tune the models on NVIDIA GPUs. Parameter-Efficient Fine-Tuning (PEFT) enhances the performance of pre-trained language models for specific tasks in Natural Language Processing. By adjusting only a subset of the model's parameters on smaller datasets, PEFT conserves computational resources and time. This method typically involves freezing several layers of the model and fine-tuning only the final layers that directly pertain to the target application, thereby achieving greater efficiency [12]. Low-Rank Adaptation (LoRA) [5] is an efficient training technique for large language models that significantly reduces the number of trainable parameters by inserting a smaller set of new weights, which are the only parts trained. This method speeds up training, enhances memory efficiency, and results in much smaller model weights (only a few hundred megabytes), making the models easier to manage and distribute. We employed the LoraConfig from the PEFT library, setting a rank of 16 and an alpha of 64 to finely tune the attention mechanism to our dataset while minimizing hardware needs without extra inference latency. In our experiment of fine-tuning the Llama-3-8b model, the original parameter count was 4,582,543,360. With LoRA, we reduced it to 41,943,040 trainable parameters, constituting just 0.915% of the total parameters. This reduction underscores the efficiency of LoRA in managing computational resources.

In our training setup, we employed a distributed system featuring eight NVIDIA GeForce RTX 3090 GPUs, leveraging PyTorch's DistributedDataParallel (DDP) framework. This choice was informed by insights from the work by Shen Li et. al (2020) which demonstrated DDP's superiority in synchronizing gradients efficiently across multiple GPUs. They noted that DDP minimizes communication overhead and optimizes computation by overlapping gradient reduction with backpropagation [23]. This

[5] https://huggingface.co/docs/diffusers/en/training/lora

results in enhanced training speed and scalability, crucial for handling large datasets and complex models in a distributed environment. This approach was particularly necessary for fine-tuning Llama models in our resource-constrained environment, where utilizing multiple GPUs for data-parallel training was essential.

Gradient accumulation [6] allows for the use of larger batch sizes than those limited by hardware memory by summing up gradients over multiple mini-batches and updating the model only after a predefined number of these batches. In our training setup, we managed a batch size of one per device, accumulating gradients over 12 steps. This strategy effectively simulates training with larger batch sizes, optimizing the trade-off between memory usage and training convergence speed.

AdamW is an optimization algorithm that modifies the classic Adam optimizer by decoupling weight decay from the gradient updates [30]. This adjustment allows for more effective and theoretically sound management of weight decay, improving generalization compared to standard weight decay in optimizers like Adam. The modification ensures that the weight decay is applied directly to the weights themselves rather than as part of the gradient descent, which helps in better preserving the training stability and often leads to better performance on validation and test datasets. We utilized the AdamW optimizer with a learning rate of 2E-4, chosen based on preliminary testing to ensure rapid convergence without compromising stability. The model underwent training over five epochs, incorporating early stopping based on validation loss to mitigate overfitting, thus enhancing model generalizability and performance efficiency.

In the code generation and execution experiment conducted on a Google Colab platform with a single NVIDIA A100 40GB GPU, the recently introduced META-LLAMA/META-LLAMA-3.1-8B model was deployed to generate Python code for financial analysis. This experiment was distinctive as it did not involve model fine-tuning. Instead, few-shot learning techniques were utilized, which included an enhanced prompt template featuring context, a specific question, and Python code examples, formatted with actual newline characters. To identify the most contextually relevant examples for the model, TF-IDF vectorization [35] and cosine similarity metrics were applied. This methodological choice was influenced by the findings of Omid et al. (2019), who observed that despite the prevalent expectation favoring advanced topic models (e.g., Latent Semantic Indexing) and neural models (e.g., Paragraph Vectors) for superior performance across all measures of textual similarity, TF-IDF demonstrated remarkable efficacy. Specifically, TF-IDF excelled in scenarios involving longer and

[6] https://huggingface.co/docs/accelerate/en/usage_guides/gradient_accumulation

more technical texts and in making more nuanced distinctions among nearest neighbors. This attribute proved particularly advantageous for the current study, which involves complex financial datasets, thereby necessitating a method capable of handling the intricacies and technicalities embedded within such texts.

Evaluation Strategy

Model performance was periodically evaluated on the validation set at the end of a predefined step (100) in the training setup.

Few-Shot Example Configuration

In Section 5.3.3, we detail the outcomes of the experiment involving few-shot prompting techniques, where we evaluated various few-shot learning methods. Few-shot learning (FSL) involves training models to recognize categories from very limited examples. One-shot learning (OSL) refers to learning tasks where only a single example per rare category is available. Zero-shot learning (ZSL), meanwhile, deals with categories for which no examples are provided [4]. Suvarna et al. (2019) discusses the challenges and techniques of generalization in few-shot learning, emphasizing that while machines need numerous examples to learn like humans, they lack certain cognitive functions. In contrast to earlier methods requiring human intervention for learning representations, modern deep learning autonomously learns these representations. However, learning effective representations from few examples remains difficult. Deep models require diverse, *representative examples* to generalize well but often struggle in few-shot scenarios due to the scarcity of such examples [13]. To tackle the challenges of few-shot learning, we developed a strategy that utilizes both relevant and random examples. The selection of relevant examples is based on cosine similarity [7] measurements between the embeddings of training samples and a predefined set of few-shot examples. These embeddings are generated using the SentenceTransformer model, specifically *all-MiniLM-L6-v2* [8] which provides a dense representation of text ideal for similarity assessments. The selection process is detailed in Algorithm 2, where TOP_K indicates the number of relevant samples calculated. In our experiments, TOP_K is set to 2 due to memory constraints and the context length limitations of the Llama-2-7b-hf and Llama-3-8b models, which are 4096 and 8192 tokens, respectively. The parameters *context_weight* and *question_weight* are both set to 0.5, reflecting the equal importance

[7] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html
[8] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

of context and questions in datasets such as ConvFinQA, where the history and the final question are crucial. This approach ensures that no input is truncated during training, although it limits our ability to test performance with a larger number of examples, i.e., a much higher TOP_K.

In the experiment for code generation detailed in section 5.4, we used few-shot learning techniques, leveraging an improved prompt template that included context, a question, and Python code examples, with a focus on proper formatting using actual newline characters. To select the most relevant examples for the model, we employed TF-IDF vectorization and cosine similarity measures. This approach ensured that the examples used in the prompt were highly pertinent to the task at hand, enhancing the model's ability to generate accurate and executable code.

Algorithm 2 Precompute Relevant Examples for Few-Shot Learning

```
function GETEMBEDDINGS(text)
    return embedding_model(**embedding_tokenizer(text, return_tensors='pt', padding=True, truncation=True)).last_hidden_state.mean(dim=1).cpu().numpy()[0]
end function
function PRECOMPUTERELEVANTEXAMPLES(dataset, few_shot_examples, top_k, context_weight, question_weight)
    few_shot_embeddings ← { "context": [GetEmbeddings(ex["context"]) for ex in few_shot_examples], "question": [GetEmbeddings(ex["question"]) for ex in few_shot_examples] }
    relevant_examples_map ← {}
    for idx, item in enumerate(dataset) do
        embeddings ← { "context": GetEmbeddings(item['context']), "question": GetEmbeddings(item['question']) }
        similarities ← { type: context_weight * cosine_similarity([embeddings[type]], few_shot_embeddings[type])[0] for type in ["context", "question"] }
        combined_similarities ← sum(similarities.values())
        most_relevant_indices ← argsort(combined_similarities)[-top_k:][::-1]
        relevant_examples_map[str(idx)] ← most_relevant_indices.tolist()
    end for
    return relevant_examples_map
end function
```

4.3.2 Post Processing Algorithm

During inference, the model generates responses based on varying prompt templates, which differ across experiments and models. Even with the same model, the prompts vary depending on whether we use few-shot prompts or additional chain-of-thought prompting to guide the model to generate explanations. Consequently, a robust post-processing algorithm is necessary. Algorithm 3 presents a generic post-processing approach used during inference and evaluation, with slight modifications based on the model's output response. The code for all post-processing steps is accessible in the corresponding GitHub repository[9].

Algorithm 3 Post-Processing Model Output

function PREPROCESSOUTPUT(output)
Input: Model's output string
Pattern Matching: Use regex to find the answer section
answer_pattern ← r"Answer (including calculation steps and final answer, use 'n' for line breaks):(.*?)
(?:n nContext:—$)"
match ← `re.search(answer_pattern, output, re.DOTALL)`
if match is not None **then**
answer ← match.group(1).strip()
lines ← [line.strip() for line in answer.split('\n') if line.strip()]
return lines[0] **if** lines **else** ""
else
return ""
end if
end function

4.4 Evaluation and Iteration

4.4.1 Permissive Accuracy

We have designed a permissive accuracy measure (as referenced in Algorithm 4) that accommodates minor precision differences by allowing slight deviations in decimal points within an alpha threshold. This measure is particularly useful in financial contexts,

[9]https://github.com/rjanant/disseration

where exact numerical precision may not always be critical. The method involves extracting numerical values and comparing them within a tunable alpha difference.

For threshold setting, we define acceptable deviation ranges, ensuring flexibility in various applications. This measure is especially relevant in scenarios like relational extraction, where generated outputs are textual values. In such cases, the SequenceMatcher is used for string comparison, returning 1 if the similarity exceeds β [10] or if key parts of the prediction and reference match. This approach is beneficial when the model generates the correct answer but includes additional values, which can be ignored. Additionally, we have an exact match function to report precise match accuracy, ensuring comprehensive evaluation of model performance.

Algorithm 4 Permissive Accuracy

function PERMISSIVE_ACCURACY(predicted, actual)
Input: predicted, actual
predicted ← `str(predicted).lower().strip()`
actual ← `str(actual).lower().strip()`
pred_num ← `ExtractNumericalValue(predicted)`
actual_num ← `ExtractNumericalValue(actual)`
if pred_num is not None and actual_num is not None **then**
Return `int(abs(pred_num - actual_num) < α)` **else**
pred_words ← `predicted.split()`
actual_words ← `actual.split()`
similarity ← `SequenceMatcher(None, pred_words, actual_words).ratio()`
key_parts_match ← `all(part in predicted for part in actual.split(':')[-1].split(','))`
Return `int(similarity > β or key_parts_match)`
end if
end function

4.4.2 Exact Match Accuracy

In the context of financial datasets, exact match accuracy is vital for ensuring the precision of numerical output generation. Given the importance of accuracy in financial

[10] $\beta = 0.9$ & $\alpha = 0.001$

analysis, even a minor deviation, such as a difference in decimal points, is unacceptable and considered an error. This metric enforces a strict criterion by directly comparing the generated output with the expected answer, ensuring only perfectly matching results are considered correct.

4.4.3 Inference Phase Metrics

4.4.3.1 Comprehensive Evaluation: Using a suite of metrics to assess various aspects of model performance

- ROUGE: It is a set of metrics and software for evaluating automatic summarization and machine translation, comparing machine outputs to human references [26].
- METEOR: Computes a score based on the harmonic mean of precision and recall, emphasizing recall, and assesses alignment of unigrams by surface, stem, and semantics, including an orderliness score [3].
- BLEU Score: Evaluates the precision of n-grams in generated text against reference [32].
- BERT Precision, Recall, F1 Score: Uses contextual embeddings to measure semantic similarity of text [48].
- Google BLEU (GLEU) Score: It modifies the traditional BLEU metric to better evaluate single sentence pairs by calculating the minimum of n-gram precision and recall between generated outputs and target sequences [45].

4.5 External Tool Integration & Few-Shot Code Generation

Recent advancements in tackling complex reasoning tasks, have opened new avenues for applying large language models to specialized domains like financial analysis. Notably, the study by Suzgun et al.(2022) provides compelling evidence for the efficacy of step-by-step approaches in solving intricate problems. The study focused on BIG-Bench, a collaborative benchmark comprising over 200 diverse text-based tasks, including traditional NLP, mathematics, commonsense reasoning, and question-answering [37].

Particularly relevant to our research is their curation of BIG-Bench Hard (BBH), a subset of 23 especially challenging tasks where previous models had not surpassed average human-rater performance.

The study's results are particularly illuminating: using various *chain-of-thought* (CoT) approaches, they found that the Codex model with CoT prompting outperformed the average human rater score on 17 out of 23 chosen tasks. This remarkable performance on diverse, complex tasks underscores the potential of step-by-step reasoning approaches in tackling challenging problems. Drawing parallels to our research, we recognized that the complex nature of financial data analysis - involving intricate calculations, temporal considerations, and the integration of tabular and textual data - could benefit from a similar methodical approach.

Inspired by these findings, our research leverages a step-by-step code generation methodology to address the unique challenges posed by financial datasets. By breaking down complex financial analyses into discrete, programmable steps, we aim to enhance the accuracy and reliability of our model's outputs. This approach allows us to handle the multifaceted nature of financial data, including numerical precision in calculations, temporal dependencies in time-series data, and the integration of qualitative information from textual sources.

Our methodology extends the concept of chain-of-thought reasoning to the domain of financial analysis, where each step in the chain is represented by a generated code snippet. This not only facilitates more accurate computations but also provides **transparency** and **interpretability** in the analysis process - crucial factors in the financial sector where decisions often require clear justification and auditable processes.

Our experimental design commenced with the careful selection of a small, handcrafted sample derived from a diverse array of financial question-answer pairs. These samples were meticulously chosen from the comprehensive dataset detailed in 4.1.1, ensuring a representative cross-section of financial queries and their corresponding responses.

To enhance the analytical capabilities of our model, we implemented a novel approach: replacing the original answers in the dataset with Python scripts. These scripts were generated with the assistance of GPT-4, a state-of-the-art language model, and subsequently underwent rigorous manual verification to ensure accuracy and functionality. This transformation allowed for a more dynamic and computationally rich representation of financial data analysis. The sample dataset was intentionally constructed to encompass a wide spectrum of answer types, including Numerical responses, Textual

answers and Temporal data. This diversity in response types was crucial for assessing the model's versatility in handling various financial data representations. For the execution of these Python scripts embedded within the answer fields, we leveraged the LangChain LLM framework integrated into our pipeline. The execution environment was facilitated by a Python agent interfacing with a Python REPL (Read-Eval-Print Loop) tool. The REPL tool was instrumental in enabling real-time execution and evaluation of Python code, a feature paramount for the dynamic and precise computation of financial metrics and analyses. The language model underpinning this integration was the FACEBOOK/OPT-1.3B MODEL, selected for its robust natural language processing capabilities and its ability to understand and generate context-aware responses. As demonstrated by Holtzman et al. (2019), Nucleus Sampling (Top-p) is highly effective for neural generation [11]. By sampling from the dynamic nucleus of the probability distribution, it balances diversity and reliability, resulting in text that mirrors human quality while maintaining fluency and coherence. In our pipeline, we use parameters: temperature=0.7, top_p=0.95, and repetition_penalty=1.15. Using these parameters means controlling the randomness of the generation (temperature=0.7), focusing on the top 95% of the probability mass (top_p=0.95), and discouraging repetitive sequences (repetition_penalty=1.15). The results are presented and analyzed in Chapter 5.

In light of the promising results obtained from the initial experiment, we proceeded to evaluate the latest META-LLAMA/META-LLAMA-3.1-8B-INSTRUCT model, a cutting-edge language model. This model demonstrated superior performance in code generation on the HumanEval benchmark, surpassing the GPT-3.5 Turbo model, and also achieved comparable results on reasoning tasks using the ARC Challenge benchmark [11], thereby establishing its suitability for our study.

The process begins with a dynamic few-shot prompting mechanism designed to optimize the model's responses to specific financial queries. To ensure contextual alignment, we utilize TF-IDF vectorization and cosine similarity to identify and select the most relevant examples from our dataset for each query. These selected examples are then integrated into an enhanced few-shot prompt template contained in A.1, along with the corresponding context and question, to guide the model in generating accurate Python code.

The code generation process is fine-tuned with parameters such as temperature (0.9) and top_p (0.95) to strike a balance between creativity and coherence. The generated code is subsequently extracted using regex pattern matching and executed within a

[11] https://ai.meta.com/blog/meta-llama-3-1/

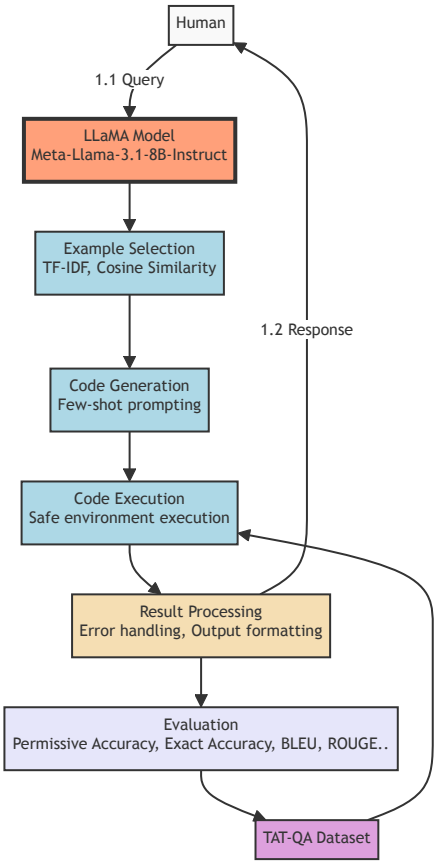

Figure 4.1: Workflow: Code Execution and Evaluation

controlled environment. This environment is equipped with the necessary libraries and contextual data, facilitating comprehensive financial data manipulation and analysis.

To evaluate the effectiveness of our approach, we processed a randomly selected subset of 1,000 samples from the TAT-QA dataset, ensuring the representativeness of the sample. For each sample, Python code was generated, executed, and the output was compared against a reference answer. The evaluation utilized the same suite of metrics detailed in Section 4.4.3, ensuring a rigorous assessment of the code generation approach within the context of financial tasks. The flowchart is explained in Figure 4.1

In summary, our code generation methodology consists of four main stages: initially, we employ few-shot prompting, supplying the models with *20* representative example samples to illustrate the desired code structure. Subsequently, the models are prompted to generate structured Python code for new inputs from the dataset. We then execute this code using Python's 'exec()' function to produce the final results. Finally, the outcomes are assessed using the diverse metrics outlined in Section 4.4.3.

Chapter 5

Experiments

This chapter delineates the framework, methodologies, and outcomes of our investigation, which focuses on enhancing the proficiency of language models in processing numerical data within financial contexts while also aiming for broad applicability across various financial tasks. We have employed a multifaceted approach that includes Instruction tuning, Few-shot prompting, Chain of Thought (CoT), and a hybrid strategy combining FSP and CoT, all designed to surpass the performance of existing models on diverse financial tasks. Additionally, we have pioneered a novel method that involves generating executable code from input data and processing it via the Python *exec* function within a structured pipeline. An exhaustive discussion of these methodologies and their implications is presented in Section 5.3, providing a comprehensive insight into the advances and innovations introduced in our study.

5.1 Research Questions

Our experiments were designed to address the following research questions:

- How do Llama2-7B and Llama3-8B, model architectures with 7 billion and 8 billion parameters respectively, differ in baseline performance on financial tasks?
- What is the relative effectiveness of various few-shot prompting strategies (e.g., zero-shot, one-shot, two-shot) compared to fine-tuning in enhancing model performance? Additionally, does the integration of Chain of Thought (CoT) techniques further influence the outcomes?
- Is it possible to fine-tune a model to perform effectively across a variety of financial tasks? If so, which types of tasks remain the most challenging?

- What is the optimal number of examples to use in few-shot prompting for financial tasks?
- How does the performance of general-purpose language models on financial data processing tasks compare to that of specialized code generation approaches?

5.2 Experimental Setup

5.2.1 Models

Our experiments were conducted using a series of progressively complex models to address the challenges posed by our dataset. Initially, we experimented with the Gemma-2b model; however, we quickly determined that its capacity was insufficient for the complexity inherent in our data, leading us to exclude it from our baseline assessments. Subsequent experiments commenced with the Llama2-7b-chat model. This decision was predicated on the observation that the base version, Llama2-7b, primarily excels in text completion tasks, while the chat variant offers enhanced flexibility through instruction tuning via system prompts. This feature allows for precise directive adjustments—for instance, directing the model to generate responses without supplementary explanations, tailored to the specific needs of our study. However, the limitations of the Llama2-7b model soon became apparent, particularly its restrictive context length, which proved inadequate for accommodating the extensive samples from our ConvFinQA dataset(see A.3.2.1 for example). Such constraints were also problematic for testing various Few-Shot Prompting (FSP) strategies, which necessitate the inclusion of additional examples in the input. Consequently, we transitioned to the Llama3-8b model for two pivotal reasons:

1. The increased context length capabilities of the Llama3-8b model allowed for a more comprehensive exploration of our dataset, which was essential for the robust evaluation of different FSP approaches which includes relevant examples in addition to the long inputs.
2. The model's enhanced parameter size and training on a more contemporary dataset meant it demonstrated superior performance across reasoning and mathematical benchmarks compared to its predecessors.

For the code generation aspect of our experiment, we selected the Llama3.1-8b variants, recognized for their superior code generation capabilities. This choice was crucial, as

our research heavily relied on the model's ability to generate precise and accurate code outputs.

Furthermore, it is pertinent to note that the fine-tuning of these models was performed using their quantized versions, the specifics of which are detailed in 4.3.1.

5.3 Results and Analysis

5.3.1 Model Architecture Comparison

Table 5.1: Evaluation Metrics for Fine tuned Models

Model	Avg Permissive Acc.	Avg Exact Acc.	ROUGE Score	BLEU Score	METEOR Score	BERTScore P	BERTScore R	BERTScore F1	GLEU Score
Llama2-7B chat baseline	0.27	0.02	0.16	0.01	0.19	0.814	0.860	0.836	0.02
Llama3-8B baseline	0.33	0.05	0.20	0.03	0.19	0.836	0.876	0.855	0.04
Llama2-7B chat Fine tuned	0.46	0.00	0.15	0.05	0.21	0.778	0.892	0.830	0.07
Llama3-8B Fine tuned	**0.67**	0.15	0.36	0.06	**0.31**	0.850	0.919	0.881	0.07
Llama3-8B - CoT	0.63	0.14	0.35	0.07	0.31	0.847	0.918	0.879	0.07
Llama3-8B - FSP+CoT (Zero shot)	0.62	**0.16**	**0.38**	**0.07**	**0.31**	**0.863**	**0.921**	**0.890**	**0.08**
Llama3-8B - FSP+CoT (One shot)	0.62	0.16	0.37	0.07	0.31	0.861	0.921	0.889	0.08
Llama3-8B - FSP+CoT (Two shot)	0.62	0.15	0.37	0.07	0.31	0.859	0.921	0.887	0.07
Llama3-8B - FSP+CoT (Random)	0.62	0.16	0.37	0.07	0.31	0.860	0.920	0.887	0.07
Llama3-8B - FSP+CoT (Mix Shot)	0.62	0.16	0.37	0.07	0.31	0.859	0.920	0.887	0.08

In the series of experiments mentioned in 5.1, we investigated the performance of two open source language models i.e. Llama2 & Llama3 across various configurations and few-shot prompting strategies, focusing on their application in our merged dataset. The analysis of Llama2-7B and Llama3-8B models revealed that the Llama3-8B model, with its additional billion parameters, exhibited a superior baseline performance. Specifically, it demonstrated a 22.2% increase in Average Permissive Accuracy and a 150% increase in Average Exact Accuracy over the Llama2-7B model. These enhancements are attributed to the increased computational capabilities and broader contextual understanding afforded by the larger model size.

Further experiments employed few-shot prompting strategies and the integration of Few Shot Prompting (FSP) & Chain of Thought (CoT) techniques. Notably, the fine-tuned Llama3-8B model showed remarkable improvement, achieving a 67% Average Permissive Accuracy, representing a substantial enhancement from its baseline performance. Moreover, the FSP+CoT Zero-shot configuration notably excelled, achieving the highest increases across several metrics including a 90.00% rise in ROUGE Score and a 133.33% increase in BLEU Score compared to the baseline Llama3-8B model. In terms of Exact Accuracy, the FSP+CoT Zero-shot approach demonstrates a 6.67% improvement over the comparably optimized fine-tuned Llama3-8B model,

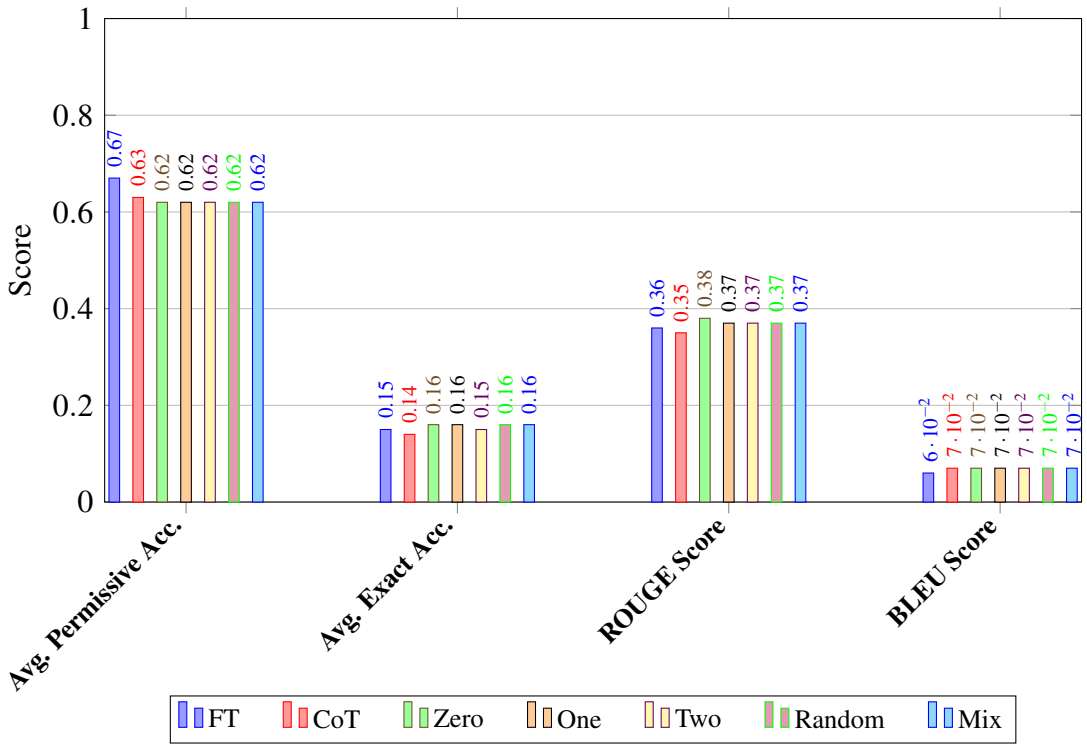

Figure 5.1: Evaluation Metrics for Selected Llama3-8b Models

underscoring its enhanced efficacy in precise response generation.

For the few-shot prompting approach, we employed the relevant examples algorithm detailed in algorithm 2. The FEW_SHOT_EXAMPLES function received 20 selectively chosen examples from the four datasets utilized in our experiments. These examples were specifically modified to differ from those used in the fine-tuning process; notably, for the TAT-QA, ConvFinQA, and FinQA datasets, the answer field was adapted to include step-by-step responses rather than a singular, final answer. In contrast, for relation extraction tasks, the examples remained akin to the training samples, focusing on direct extraction from provided information without necessitating stepwise reasoning. It is pertinent to highlight that all experimental results reported in 5.1 were derived from testing on a comprehensive dataset comprising 1287 samples, which collectively encompass elements from all four datasets involved in the study.

In addressing our second research question regarding the efficacy of various few-shot prompting strategies and the potential enhancement provided by the Chain of Thought (CoT) technique, our analysis yielded intriguing results. Interestingly, it is observed that CoT approach independently couldn't perform as good as when it is used in combination with different few shot approaches. Contrary to expectations, we observed no substantial performance differences when the model was prompted

with either random or relevant examples, or even without any specific examples (zero-shot). This lack of differential suggests a performance plateau from tailored example inputs, which might be attributed to several factors: potential saturation in the model's learning capability from few-shot examples, the model's high baseline abilities resulting from extensive pre-training, and the complex nature of financial tasks which may require more than mere contextual cues provided by few-shot examples. Regarding the integration with CoT, our results demonstrated noticeable enhancements in Exact Match Accuracy along with other critical metrics, including ROUGE, BLEU, METEOR, BERT, and GLEU Scores. These improvements establish CoT in conjunction with zero shot prompting as the superior approach, notwithstanding its marginally lesser performance in terms of Permissive Accuracy. These metrics are crucial for evaluating the model's performance as they assess the congruence between the predicted responses and the references, encompassing a variety of aspects. This is particularly pertinent given that our generated responses are not solely numerical but also include a significant proportion of textual answers, especially prevalent within the relation extraction dataset.

The decision to explore various few-shot strategies, despite the similar outcomes, was driven by the intent to rigorously test the model's adaptability to different instructional contexts. This was essential to identify any potential nuances in how different prompting strategies might affect performance. While these strategies did not yield the expected variance in results, they provided valuable insights into the limitations and capabilities of current LLMs in handling complex tasks like financial analysis.

5.3.2 Few-shot Prompting on individual datasets

In this analysis, we conduct a detailed examination of the Few-Shot Prompting approach, assessing its performance across individual datasets and corresponding metrics. It is critical to emphasize that the evaluations for each dataset were performed on their respective test sets, ensuring that the results presented are dataset-specific and not generalized across all datasets included in this study. The model employed for inference in this study was consistent with the one utilized in prior experiments; specifically, it was fine-tuned using the Chain of Thought (CoT) and Few-Shot Prompting (FSP) approaches. This fine tuned model utilises either 0, 1 or 2 relevant examples, which were selected randomly to optimize its performance and applicability to the tasks at hand.

The results from the TAT-QA dataset indicated that while there was a slight im-

Table 5.2: Evaluation Results on TAT-QA dataset

Approach	Avg Permissive Acc.	Avg Exact Acc.	ROUGE Score	BLEU Score	METEOR Score	BERTScore P	BERTScore R	BERTScore F1	GLEU Score
Zero Shot	0.59	**0.14**	**0.43**	**0.12**	**0.30**	**0.869**	**0.922**	**0.894**	**0.12**
One Shot	**0.60**	0.13	0.41	0.11	0.29	0.863	0.920	0.889	0.11
Two Shot	**0.60**	0.12	0.39	0.11	0.29	0.859	0.918	0.886	0.10
Random Shot	0.59	0.13	0.41	0.11	0.29	0.863	0.919	0.889	0.11
Mix Shot	0.59	0.13	0.40	0.11	0.29	0.860	0.918	0.887	0.10

Table 5.3: Evaluation Results on ConvFinQA Dataset

Approach	Avg Permissive Acc.	Avg Exact Acc.	ROUGE Score	BLEU Score	METEOR Score	BERTScore P	BERTScore R	BERTScore F1	GLEU Score
Zero Shot	**0.68**	**0.56**	**0.78**	0.00	0.28	**0.977**	0.975	**0.976**	0.27
One Shot	**0.68**	**0.56**	**0.78**	0.00	0.28	0.976	0.975	**0.976**	**0.34**
Two Shot	0.67	**0.56**	0.77	0.00	0.28	0.976	0.975	0.975	0.31
Random Shot	**0.68**	**0.56**	**0.78**	0.00	0.28	**0.977**	0.975	**0.976**	0.27
Mix Shot	**0.68**	**0.56**	**0.78**	0.00	0.28	0.976	0.975	0.975	0.32

Table 5.4: Evaluation Results on FinQA Dataset

Approach	Avg Permissive Acc.	Avg Exact Acc.	ROUGE Score	BLEU Score	METEOR Score	BERTScore P	BERTScore R	BERTScore F1	GLEU Score
Zero Shot	**0.30**	**0.02**	**0.12**	0.00	**0.10**	**0.810**	0.806	0.803	0.01
One Shot	0.28	**0.02**	0.11	0.00	0.09	**0.810**	0.806	**0.804**	0.01
Two Shot	0.28	**0.02**	0.10	0.00	0.09	0.806	**0.807**	0.801	0.01
Random Shot	0.29	**0.02**	0.10	0.00	0.09	0.804	0.806	0.800	0.01
Mix Shot	0.29	**0.02**	0.11	0.00	**0.10**	0.808	**0.807**	0.802	0.01

Table 5.5: Evaluation Results on Relation-Extraction Dataset

Approach	Avg Permissive Acc.	Avg Exact Acc.	ROUGE Score	BLEU Score	METEOR Score	BERTScore P	BERTScore R	BERTScore F1	GLEU Score
Zero Shot	**0.85**	0.00	**0.19**	**0.05**	**0.40**	**0.817**	**0.899**	**0.856**	0.09
One Shot	**0.85**	0.00	**0.19**	**0.05**	0.39	**0.817**	0.898	0.855	0.09
Two Shot	0.84	0.00	**0.19**	**0.05**	0.39	**0.817**	0.898	0.855	0.09
Random Shot	**0.85**	0.00	**0.19**	**0.05**	**0.40**	**0.817**	**0.899**	**0.856**	0.09
Mix Shot	0.84	0.00	**0.19**	**0.05**	0.39	**0.817**	0.898	0.855	0.09

provement with increased prompting, the gains were marginal, pointing to a potential plateau in performance enhancement achievable through few-shot methods. This was particularly evident as the highest Average Permissive Accuracy reached was 0.60 for one and two-shot scenarios. However, the consistently low Exact Accuracy rates across these configurations underscored significant challenges in achieving precise numerical reasoning. These observations are consistent with earlier results 5.1, which highlighted that the Zero Shot performance generally yielded the best outcomes when dealing with data that integrates tabular content with textual data, thereby reinforcing the effectiveness of the Chain of Thought (CoT) approach regardless of the use of examples. In the ConvFinQA dataset, the models demonstrated robust performance, consistently achieving high Permissive Accuracies around 0.68 and Exact Accuracies at 0.56 across most approaches. This dataset, which benefits from not truncating inputs despite longer context lengths, showcased our tuned models' strong performance in

understanding QA history—a task generally considered challenging. Notably, these results did not significantly differ across the few-shot prompting variations, which highlighted a potential limitation in the impact of adding more contextual examples within datasets well-aligned with the models' training. Additionally, an unexpected observation was the consistent 0 scores for the BLEU Score metric in both the ConvFinQA and FinQA datasets. Upon further investigation, it was revealed that these scores reflect the limitations of applying the BLEU score metric, traditionally designed for corpus-level evaluation, to single predictions and references. As Wu et al. (2016) identified, the BLEU score exhibits several undesirable properties when utilized for single sentences [45]. Consequently, we have also reported the GLEU Score, which is designed to mitigate these undesirable properties when assessing single sentences. This adjustment provides a more appropriate measure of accuracy in contexts where traditional metrics such as BLEU may not be applicable.

The FinQA dataset presented considerable challenges, with Permissive Accuracies barely under 0.30 and negligible Exact Accuracy. This dataset, based on earnings reports from SP 500 companies, necessitates a high degree of inferential reasoning. For example, questions such as *Considering the weighted average fair value of options, what was the change of shares vested from 2005 to 2006?* require the model to calculate the number of shares and select relevant numbers from both the table and the text to generate the reasoning program to arrive at the answer [6], which remains a substantial hurdle, indicating the model's shortcomings in complex text understanding and information extraction from dense financial narratives. To answer our third research question, we can say that although the model demonstrated a 6.67% improvement over the fine-tuned Llama-3-8b model in terms of exact match on the merged test set, thus improving performance across diverse financial tasks, the FinQA questions remain the most challenging for the model 5.4. Further analysis and research are needed to enhance the model's performance on this specific dataset.

The findings from the Relation-Extraction dataset unveiled a pronounced discrepancy between high Permissive Accuracy and zero Exact Accuracy, illustrating a critical challenge in the model's ability to execute precise relation extraction. Although the model demonstrated competency in broadly identifying correct relational concepts, it faltered in accurately pinpointing and extracting the most relevant relations. This challenge is exemplified by the model's response to structured prompts designed to extract the single most prominent relation. For instance, given the context from a news conference by JPMorgan Chase Co's CEO, Jamie Dimon, the model was instructed to

identify specific relationships within the sentence. Despite the intended response being *employer: Jamie Dimon, JPMorgan Chase*, the model provided an overly comprehensive array of relational data, including multiple unrelated relations such as *position held: Jamie Dimon, Chief Executive Officer* among others (see A.3.1.3 for a complete example). This overflow of information suggests that while the model can recognize relational data, it tends to overgeneralize from the input, leading to responses that extend beyond the required extraction. This issue could potentially be mitigated by refining the model's prompt structure to more clearly delineate the expected response boundaries, yet the dataset inherently demands multiple extractions per instance, complicating the clarity of how many relations are to be extracted. This complexity necessitated the introduction of an additional metric, Average Permissive Accuracy, to provide a more lenient measure of the model's performance, acknowledging that while not exactly precise, the model successfully captures the correct answers a significant portion of the time. Despite this adaptation, we continue to report the Exact Match Accuracy to maintain a rigorous standard of evaluation and to transparently assess the model's precision in relation extraction tasks.

Revisiting the research question on the optimal number of examples for Few-Shot Prompting (FSP) in financial tasks, our findings reveal that increasing the number of examples does not consistently lead to significant performance gains across all datasets. This suggests that the quality and relevance of examples, particularly in designing prompts that closely reflect task-specific requirements, may be more crucial than merely increasing the number of examples provided. Furthermore, our experiments indicate that the Chain-of-Thought (CoT) approach performs effectively even with zero shots, *suggesting that the combination of FSP with examples and CoT is not always necessary, particularly for complex tasks where the model tends to rely more on training data than on a few new examples provided during inference*. In cases where few-shot prompting did not enhance performance, it is likely that the complexity and specificity of the tasks diminish the benefits of additional examples, highlighting an important area for future research.

5.3.3 Tool-based Approach

The tool-based approach using Langchain on a subset of 150 samples showed promising results:

- Exact Accuracy: 0.75

- BLEU Score: 0.23
- METEOR Score: 0.33

The findings from this study underscore the efficacy of specialized tools in performing precise financial tasks, with a notable requirement for exact matches in data handling. It is pertinent to acknowledge that these results are derived from scenarios utilizing manually generated code, specifically using GPT-4 [1], which represents a potentially idealized set of conditions.

An interesting continuation of this research could involve leveraging OpenAI's advanced code generation capabilities in conjunction with Langchain's Python agent REPL tool to automate and refine this process further. However, such an approach would entail additional costs, which were not within the purview of this current study. Nevertheless, the potential of commercial language models to adeptly manage the complexity and variability inherent in our financial dataset remains a compelling aspect of our ongoing investigation. This line of inquiry suggests substantial promise for enhancing automated financial analysis tools through the integration of sophisticated AI-driven technologies.

5.4 Code Generation Approach

In this series of experiments, we investigate an innovative approach to enhancing numerical precision in language models applied to financial tasks through the integration of code generation and execution. Addressing queries based on financial reports, which often encompass both tabular and textual data (hybrid data), poses significant challenges. These tasks require models not only to extract relevant information from financial documents but also to conduct complex quantitative analyses. While current models have shown strong capabilities in answering straightforward questions, they often struggle with more complex queries that necessitate multi-step numerical reasoning. Liu et al. (2024) observed that language models designed for code synthesis are systematically trained to correlate natural language specifications with coding patterns derived from extensive datasets, ranging from thousands to millions of examples. This training strategy inherently limits their ability to perform deep code reasoning, thereby reducing their effectiveness in addressing complex, real-world programming challenges and extensive program analysis tasks [27]. Several prior studies have sought to enhance the numerical reasoning abilities of language models. For instance, Li et

al. (2022) introduced a framework called FinMath, which enhances a model's capacity for numerical reasoning by integrating a tree-structured neural model to perform multi-step calculations using supporting evidence from financial reports [20]. Although numerous attempts have been made to improve the handling and reasoning capabilities of language models, especially with hybrid data, the specific application of code generation for complex financial tasks remains relatively unexplored. Kang et al. (2024) demonstrated the use of language models to generate application code automating processes in health insurance based on text-based policies [14]. Our approach harnesses the structured logic inherent in programming languages to enforce logical consistency in these tasks. Previous research has established the efficacy of step-by-step methodologies when dealing with hybrid data, which often requires intricate calculations, arithmetic operations, and reasoning skills. This approach mirrors the manual coding process used to solve similar hybrid tasks: extracting variables (when table extraction is necessary), performing operations on these variables, and ultimately calculating and presenting the final result. Building on this concept, as well as promising results from a tool-based approach that demonstrated high exact match accuracy using a small subset of data, we conducted our experiments. In our experiments, we utilized carefully selected examples of Python scripts tailored for financial calculations. The proposed pipeline, which comprises code generation, execution, and post-processing, is designed to facilitate modular improvements and allow for error handling at each stage. The code was executed within a secure and controlled environment, addressing potential security concerns while enabling dynamic code execution. We argue that this methodology bridges the gap between natural language processing and programmatic solutions, offering the potential for more precise and verifiable results compared to purely NLP-based approaches.

5.4.1 Experimental Setup

For these experiments, we utilized the TAT-QA dataset, a benchmark for text-and-table question answering [51]. To ensure a manageable yet representative sample, we randomly selected 1,000 samples from the dataset, maintaining consistency across all models by using a fixed seed value for reproducibility. Our experiments were conducted using several models from the Llama family, including Llama3-8b, Llama3.1-8b, and Llama3.1-8b-instruct. It's important to note that the Llama models, especially the 8b variants used in this study, are not primarily known for their code generation

capabilities. This selection was deliberate, allowing us to test the limits of our approach and potentially demonstrate its efficacy even with models not specialized for this task.

5.4.2 Results and Discussion

Table 5.6: Evaluation Metrics for Code Generation

Model	Avg Permissive Acc.	Avg Exact Acc.	ROUGE Score	BLEU Score	METEOR Score	BERTScore P	BERTScore R	BERTScore F1	GLEU Score
Llama3-8b	0.39	0.30	0.40	0.18	0.32	0.870	0.898	0.884	0.19
Llama3.1-8b	0.40	0.30	0.39	0.17	0.32	0.868	0.898	0.882	0.18
Llama3.1-8b-instruct	**0.52**	**0.43**	**0.49**	**0.29**	**0.34**	**0.887**	**0.909**	**0.897**	**0.24**
Llama3.1-8b (**code-gen**)	0.23	0.18	0.25	0.05	0.14	0.841	0.874	0.856	0.06
Llama3.1-8b-instruct (**code-gen**)	0.35	0.28	0.41	0.14	0.21	0.866	0.898	0.881	0.14

Table 5.7: Filtered sampling in Code Generation

Model	Total samples selected	Remaining samples	Percentage of samples with execution errors	Utilization Rate
Llama3.1-8b (**code-gen**)	1000	779	31.58	53.3
Llama3.1-8b-instruct (**code-gen**)	1000	843	7.12	78.3

In our preliminary investigations, we explored the application of the CodeLlama-7b-Python-hf model for code generation, inspired by the research conducted by Yang et al. (2024). Their study illustrated the efficacy of Large Language Models (LLMs) in generating Prolog code to solve arithmetic problems, demonstrating a significant superiority over chain-of-thought (CoT) approaches in a variety of models including Llama-2, CodeLlama [33], and Mistral within the GSM8K benchmark [47]. Despite these encouraging results, our experiments using the CodeLlama-7b-Python variant with the hybrid TAT-QA dataset indicated significant performance deficits. The model exhibited a high execution error rate of 96%, primarily due to difficulties in processing the complex structures and instructions embedded within the dataset. This outcome necessitated a reassessment of the effectiveness of smaller parameter models under computational constraints, particularly in handling intricate datasets such as TAT-QA. Consequently, these insights led to a pivotal redirection in our research methodology.

In response to the limitations observed with the CodeLlama-7b model, we undertook a fine-tuning process on base models, specifically targeting the Llama-3-8b, Llama-3.1-8b, and Llama-3.1-8b-instruct variants. Our choice to engage the instruct variant stemmed from observations of the conventional non-instruct models, which frequently attempted to access non-existent CSV files—a behavior likely ingrained from their training on diverse datasets that routinely involved importing and processing tabular data from external sources. In contrast, our instruction-tuned models were meticulously

crafted to operate solely within the given context, generating Python code exclusively based on the provided inputs. This choice was also influenced by the use of Few Shot Prompting, as we were not fine-tuning the models but directly employing them, necessitating the model's adherence to instructions and examples provided. The results, as shown in 5.6, indicate that the Llama-3.1-8b-instruct model consistently outperformed its non-instruct counterpart across all metrics. For example, in the code generation task, the instruct model achieved a 52% higher average permissive accuracy (0.35 vs. 0.23) and a 55% higher average exact accuracy (0.28 vs. 0.18) compared to the non-instruct model. Interestingly, although we anticipated the code generation approach to outperform direct output, the direct output method surpassed code generation for both Llama-3.1-8b and Llama-3.1-8b-instruct. This suggests that while code generation shows promise, improvements are needed in the quality and reliability of the generated code. Comparatively, the smaller FACEBOOK/OPT-1.3B model, when using Langchain, exhibited impressive performance on its own test set, achieving a 0.75 exact accuracy. However, this result was based on a much smaller sample size (150) compared to our main experiments (1000). Our approach, however, did not surpass the performance of directly generated responses or traditional methods. Further investigation into this revealed notable observations, as detailed in 5.7. In our experiment for Python script generation, we intentionally filtered out samples that did not generate any code, while keeping track of these samples as well as the percentage where code generation led to execution errors. The Llama-3.1-8b-instruct model demonstrated marked improvements in code generation reliability, with fewer samples filtered out (157 vs. 221) and a significantly lower percentage of execution errors (7.12% vs. 31.58%) compared to the non-instruct model. Overall, the non-instruct variant failed on nearly 47% of samples in code generation, whereas the instruct variant failed on about 22% of samples. We then calculated a utilization rate defined as the proportion of usable samples after filtering and errors, relative to the overall count. The utilization rate U is expressed mathematically as:

$$\mathrm{U} = \left(\frac{n_{\text{filtered}}}{n_{\text{total}}} \right) \times 100\%$$

The instruct model achieved a much higher utilization rate (78.3% vs. 53.3%), indicating its greater success in generating and executing code for a larger proportion of input samples. To estimate the metrics if all samples had been processed successfully, we scaled the observed scores proportionally to the total sample size before any filtering or errors, computing results for both models. The scaling factors for these models were

calculated to be 1.877 and 1.277, respectively. Using these scaling factors, we estimated the metrics as if the entire sample set of 1000 could be used with similar performance. The estimated average permissive accuracy was 0.430 for Llama-3.1-8b (code gen) and 0.447 for Llama-3.1-8b-instruct (code gen), which is higher than the Llama-3.1-8b model generating direct responses. In terms of exact match accuracy, the scaling factors suggested results of 0.340 for Llama-3.1-8b (code gen) and 0.357 for Llama-3.1-8b-instruct (code gen), again outperforming the Llama-3.1-8b model, although still lagging behind the instruct base variant in comparison. It is important to note that this scaling represents a best-case scenario and may not reflect the real-world distribution of the entire dataset. Other metrics, such as ROUGE, BLEU, METEOR, BERT, and GLEU scores, reported in the table imply similar observations, with higher scores for instruct models indicating better overlap with the generated response. However, the code generation approach consistently lagged behind in these metrics as well.

Revisiting our research question on the comparative performance of language models versus specialized code generation approaches, the results reveal significant challenges presented by the TAT-QA dataset, particularly for the relatively smaller 8-billion-parameter models used in this study. Initially, we observed a high rate of filtered samples and frequent generation of erroneous code, which can be attributed to the complexity of handling hybrid datasets comprising both tabular and textual data with extensive context inputs. The complexity is further exacerbated by the inclusion of diverse data types and symbols, such as currency symbols ($), years, dates, Roman numerals, tags (e.g., ‘\n‘, ‘\t‘), floating-point values, and currency conversions between units like dollars and pounds, as well as value conversions to millions and billions(sample response from the model - A.3.4). These factors present considerable challenges for language models in accurately comprehending and extracting relevant information from tables or text. Although larger models demonstrate improved performance in these tasks, generating precise Python scripts from such complex datasets remains challenging. We hypothesize that utilizing larger models, or models specifically designed for code generation, given their capacity to process extensive and complex input tokens, may potentially yield more accurate results, surpassing the performance of the baseline models employed in this study. Furthermore, while fine-tuning the model could potentially enhance its performance, this would necessitate a large dataset containing correct Python scripts for all samples, which is currently unavailable for our task involving financial data.

Chapter 6

Conclusion

Our research investigates enhancements for open-source language models in financial domain. Despite recent advancements, LLMs like GPT-4o are yet to match human expertise in numerical reasoning within extensive contexts, often struggling with complex instructions involving multiple tasks or noisy, heterogeneous inputs [49, 10]. Additionally, while these models exhibit impressive in-context learning abilities, they fall short in relation extraction compared to supervised methods, hindered by ineffective demonstration retrieval and limited in-context application [21]. Our study underscores that significant improvements in the handling of numerical data for financial tasks can be achieved through the deployment of advanced model architectures, targeted fine-tuning, and the strategic use of few-shot prompting (FSP). Notable improvements were observed with the Llama-3-8b FSP+CoT (Zero shot) model, which combined FSP with Chain-of-Thought (CoT) approaches. This model, tested on a merged dataset, demonstrated an approximate 87.8% increase in permissive accuracy and a substantial 220% increase in exact match accuracy over the baseline Llama-3-8b model. Interestingly, despite the prevalent view that FSP and CoT typically enhance LLM performance, our findings reveal that these methods did not consistently yield superior results. Instead, the fine-tuned 8b model exhibited better performance in terms of average permissive accuracy, suggesting that the complexity of financial tasks may diminish the effectiveness of adding numerous relevant examples during inference. An unexpected observation from our study was the inefficacy of increasing the number of relevant examples, which in some instances performed similarly to or even worse than randomly selected examples on certain metrics. This finding leads us to hypothesize that in complex tasks, a model that is meticulously trained with tailored instructions may rely more on structured guidance rather than on multiple example prompts.

The methodology adopted in our research, which focuses on code generation and execution for financial applications, represents a pioneering endeavor in this field. The approach yielded fascinating outcomes, though it confronted numerous obstacles related to managing hybrid datasets and the subsequent generation of code. Notably, while the specialized code-generating model, CodeLlama-7b-Python, is generally recognized for its superior performance in code generation compared to its baseline variants, it proved inadequate for our dataset-specific challenges. Our study leveraged the code generation capabilities inherent in language models to enhance numerical precision in financial tasks, despite challenges with code reliability. By incorporating instruction tuning, our approach achieved a 55% performance improvement over non-instruct models, with fewer execution errors and better utilization rates. However, the limitations of our methodology become evident when juxtaposed with traditional direct response strategies. Despite these caveats, our findings affirm the viability of this innovative approach. When implemented with an optimally tuned model that minimizes code execution errors, it has the potential to surpass the outcomes of direct response methodologies.

6.1 Future Work

Although our study was limited by computational resources, allowing us to test with a maximum of two examples per prompt due to constraints, future work could explore whether increasing this number to four or five might alter the outcomes. Such investigations could potentially demonstrate that a greater volume of contextual data might more effectively steer model performance. This prospective area of research will aim to further dissect the nuanced applications of LLMs in specialized domains, where the intricate nature of tasks could overshadow the traditional benefits of model enhancements. Future research emerging from our initial investigations holds the potential to profoundly impact code generation and execution in numerically demanding domains. A significant aspect of this future work is the advancement and implementation of larger, domain-specific models tailored to meet the distinct demands of sectors such as finance or related fields. This would involve refining few-shot prompting techniques to bolster code generation reliability, potentially incorporating adaptive prompting mechanisms that evolve through feedback loops, as exemplified by the GUIDED EVOLUTION framework. This framework uses Large Language Models (LLMs) to supervise mutations and optimize code evolution, reflecting a new direction in model adaptability [31].

Bibliography

[1] OpenAI (2023). Gpt-4 technical report, 2024.

[2] Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models, 2019.

[3] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[4] Preston Billion Polak, Joseph D Prusa, and Taghi M Khoshgoftaar. Low-shot learning and class imbalance: a survey. *Journal of Big Data*, 11(1):1, 2024.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] 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. Finqa: A dataset of numerical reasoning over financial data. *Proceedings of EMNLP 2021*, 2021.

[7] Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. Convfinqa: Exploring the chain of numerical reasoning in conversational finance question answering, 2022.

[8] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[10] Qianyu He, Jie Zeng, Wenhao Huang, Lina Chen, Jin Xiao, Qianxi He, Xunzhe Zhou, Jiaqing Liang, and Yanghua Xiao. Can large language models understand real-world complex instructions? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18188–18196, 2024.

[11] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[12] Mathav Raj J, Kushala VM, Harikrishna Warrier, and Yogesh Gupta. Fine tuning llm for enterprise: Practical guidelines and recommendations, 2024.

[13] Suvarna Kadam and Vinay Vaidya. Review and analysis of zero, one and few shot learning approaches. In *Intelligent Systems Design and Applications: 18th International Conference on Intelligent Systems Design and Applications (ISDA 2018) held in Vellore, India, December 6-8, 2018, Volume 1*, pages 100–112. Springer, 2020.

[14] Inwon Kang, William Van Woensel, and Oshani Seneviratne. Using large language models for generating smart contracts for health insurance from textual policies, 2024.

[15] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks, 2023.

[16] Seungone Kim, Se June Joo, Doyoung Kim, Joel Jang, Seonghyeon Ye, Jamin Shin, and Minjoon Seo. The cot collection: Improving zero-shot and few-shot learning of language models via chain-of-thought fine-tuning, 2023.

[17] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023.

[18] Jean Lee, Nicholas Stevens, Soyeon Caren Han, and Minseok Song. A survey of large language models in finance (finllms), 2024.

[19] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.

[20] Chenying Li, Wenbo Ye, and Yilun Zhao. FinMath: Injecting a tree-structured solver for question answering over financial reports. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 6147–6152, Marseille, France, June 2022. European Language Resources Association.

[21] Guozheng Li, Peng Wang, Wenjun Ke, Yikai Guo, Ke Ji, Ziyu Shang, Jiajun Liu, and Zijie Xu. Recall, retrieve and reason: Towards better in-context relation extraction, 2024.

[22] Junyi Li, Jie Chen, Ruiyang Ren, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. The dawn after the dark: An empirical study on factuality hallucination in large language models, 2024.

[23] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020.

[24] Xianzhi Li, Samuel Chan, Xiaodan Zhu, Yulong Pei, Zhiqiang Ma, Xiaomo Liu, and Sameena Shah. Are chatgpt and gpt-4 general-purpose solvers for financial text analytics? a study on several typical tasks, 2023.

[25] Yinheng Li, Shaofei Wang, Han Ding, and Hang Chen. Large language models in finance: A survey. In *Proceedings of the fourth ACM international conference on AI in finance*, pages 374–382, 2023.

[26] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[27] Changshu Liu, Shizhuo Dylan Zhang, Ali Reza Ibrahimzada, and Reyhaneh Jabbarvand. Codemind: A framework to challenge large language models for code reasoning, 2024.

[28] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is

better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

[29] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing, 2021.

[30] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[31] Clint Morris, Michael Jurado, and Jason Zutty. Llm guided evolution - the automation of models advancing models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '24, page 377–384, New York, NY, USA, 2024. Association for Computing Machinery.

[32] Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. Bleu: a method for automatic evaluation of machine translation. pages 311–318, 2002.

[33] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

[34] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023.

[35] Omid Shahmirzadi, Adam Lugowski, and Kenneth Younge. Text similarity in vector space models: A comparative study. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 659–666, 2019.

[36] Karan Singhal, Shekoofeh Azizi, Tao Tu, S. Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, Perry Payne, Martin Seneviratne, Paul Gamble, Chris Kelly, Nathaneal Scharli, Aakanksha Chowdhery, Philip Mansfield, Blaise Aguera y Arcas, Dale Webster, Greg S. Corrado, Yossi Matias, Katherine Chou, Juraj Gottweis, Nenad

Tomasev, Yun Liu, Alvin Rajkomar, Joelle Barral, Christopher Semturs, Alan Karthikesalingam, and Vivek Natarajan. Large language models encode clinical knowledge, 2022.

[37] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[39] Neng Wang, Hongyang Yang, and Christina Dan Wang. Fingpt: Instruction tuning benchmark for open-source large language models in financial datasets, 2023.

[40] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023.

[41] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions, 2023.

[42] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2022.

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

[44] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance, 2023.

[45] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff

Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[46] Haoyi Xiong, Jiang Bian, Sijia Yang, Xiaofei Zhang, Linghe Kong, and Daqing Zhang. Natural language based context modeling and reasoning for ubiquitous computing with large language models: A tutorial, 2023.

[47] Xiaocheng Yang, Bingsen Chen, and Yik-Cheung Tam. Arithmetic reasoning with llm: Prolog generation permutation, 2024.

[48] Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*, 2020.

[49] Yilun Zhao, Yitao Long, Hongjun Liu, Ryo Kamoi, Linyong Nan, Lyuhao Chen, Yixin Liu, Xiangru Tang, Rui Zhang, and Arman Cohan. Docmath-eval: Evaluating math reasoning capabilities of llms in understanding long and specialized documents, 2024.

[50] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *International conference on machine learning*, pages 12697–12706. PMLR, 2021.

[51] Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 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, August 2021. Association for Computational Linguistics.

[52] Fengbin Zhu, Ziyang Liu, Fuli Feng, Chao Wang, Moxin Li, and Tat-Seng Chua. Tat-llm: A specialized language model for discrete reasoning over tabular and textual data, 2024.

Appendix A

A.1 Prompt Templates

LLAMA2-7B-CHAT-HF Prompt

```
<s>[INST]
<<SYS>>
{system_prompt}
<</SYS>>
{user_message} [/INST]
```

System Prompt: "You are a helpful AI assistant specializing in financial analysis. Answer the following question based on the given context."

LLAMA3-8B Prompt

```
Below is an instruction that describes a task,
paired with an input that provides further context.
Write a response that appropriately completes the request.
### Instruction:
{}
### Input:
{}
### Response:
{}
```

LLAMA3-8B WITH FSP+CoT Prompt

```
You are a financial expert assistant. Answer the following question
based on the given context. It is CRUCIAL that you use
step-by-step reasoning and provide detailed numerical calculations
    where applicable.
Break down your answer into clear, numbered steps.
If asked to extract subject and object in relation, return only the
    single
most relevant and applicable extraction. Use '\n' for line breaks in
    your response.
Examples:
{examples}
Now, please answer the following question using the same step-by-step
approach as demonstrated in the Examples:
Context: {context}
Question: {question}
Answer (including calculation steps and final answer, or single
most relevant relation extraction, use '\n' for line breaks):"""
```

Improved Few Shot Prompt for Code Generation and Execution

Below is an instruction that describes a task, paired with examples and an input that provides further context. Learn from the examples and write a response that appropriately completes the request.

Instruction: You are an AI assistant specialized in financial analysis. Your task is to generate Python code that answers questions based on given financial data and context. Learn from the examples provided and generate accurate, executable Python code that solves the given problem.

Important: When writing Python code, use actual newline characters to separate lines, not ' n' string literals. Each line of code should be on its own line in your response.

Examples: {examples}

Input: Context: {context} **Question:** {question}

Response: Here's the Python code to answer the question: ```python

A.2 Samples

A.2.1 TAT-QA Dataset

Context: Actuarial assumptions The Group\u2019s scheme liabilities are measured using the projected unit credit method using the principal actuarial assumptions set out below: Notes: 1 Figures shown represent a weighted average assumption of the individual schemes. 2 The rate of increases in pensions in payment and deferred revaluation are dependent on the rate of inflation.\nTable:\n \t2019 \t2018 \t2017 \nWeighted average actuarial assumptions used at 31 March1:\t\t\t\nRate of inflation2 \t2.9\t2.9 \t3.0 \nRate of increase in salaries \t2.7 \t2.7 \t2.6 \nDiscount rate \t2.3 \t2.5 \t2.6

Question 1: What does the Weighted average actuarial assumptions consist of?

Answer: Rate of inflation, Rate of increase in salaries, Discount rate

Question 2: How much is the 2019 rate of inflation?

Answer: 2.9

A.2.2 Relation Extraction Dataset

Context: LUXEMBOURG, July 9 National parliaments should be convinced to vote in favour of helping Greece if European leaders can reach a good agreement on Sunday, President of the European Council Donald Tusk said, adding that a deal on debt should be part of the agreement.

Question : Given the input sentence, please extract the subject and object containing a certain relation. Relations include: product/material produced; manufacturer; distributed by; industry; position held; original broadcaster; owned by; founded by; distribution format; headquarters location; stock exchange; currency; parent organization; chief executive officer; director/manager; owner of; operator; member of; employer; chairperson; platform; subsidiary; legal form; publisher; developer; brand; business division; location of formation; creator.

Answer: position_held: Donald Tusk, President of the European Council

A.2.3 ConvFinQA Dataset

Note: This sample is summarized for clarity and ease of interpretation.
In fiscal 2008, revenues in the credit union systems and services business segment increased by 14% from fiscal 2007. All revenue components within the segment experienced growth. The largest dollar growth was noted in license revenues from Episys AE, our flagship core processing system aimed at larger credit unions, which experienced strong sales throughout the year. Support and service revenue, the largest component of total revenues for the credit union segment, experienced 34% growth in EFT support and 10% growth in in-house support. Gross profit in this business segment increased by $9344 in fiscal 2008 compared to fiscal 2007, due primarily to the increase in license revenue, which carries the highest margins. liquidity and capital resources we have historically generated positive cash flow from operations and have generally used funds generated from operations and short-term borrowings on our revolving credit facility to meet capital requirements . we expect this trend to continue in the future . the company 2019s cash and cash equivalents increased to $ 118251 at june 30 , 2009 from $ 65565 at june 30 , 2008 . the following table summarizes net cash from operating activities (NCOA) in the statement of cash flows:

Year	Net Income ($)	Non-Cash Expenses ($)	NCOA($)
2009	104681	56348	206588
2008	104222	70420	181001
2007	103102	74397	174247

In 2009, cash used in investing activities was $59227, including $3027 in contingent consideration paid on prior years' acquisitions. Capital expenditures were $31562, compared to $31105 in 2008. Net cash used in financing activities was $94675, including the repurchase of shares and payment of dividends. Question: What is the net cash from operating activities in 2009? Answer: 206,588
Question: what about in 2008?
Answer: 181001.0

A.2.4 FinQA Dataset

Note: This sample is summarized for clarity and ease of interpretation.

Financial Instruments and Risk Management Summary:

Contingent Liabilities: Adjustments for contingent liabilities related to business acquisitions are made until they are settled.

Derivatives and Hedging: The company utilizes interest rate swaps to hedge its variable rate debts, treating them as cash flow hedges in compliance with FASB ASC Topic 815. These derivatives are effectively managing interest rate risks without affecting earnings, maintaining strict policies against speculative use. Foreign exchange risks are mitigated through forward contracts, notably against the INR, to stabilize operational outcomes against currency fluctuations.

Trade Receivables: As of the end of 2016 and 2015, the company reported net trade receivables of $1,639 million and $1,731 million, respectively. These figures account for allowances for doubtful accounts and adjustments made annually.

Significant Transactions in 2015-2016: The company settled treasury lock hedges with a notional amount of $1.0 billion in October 2015, relating to a cash settlement payment of $16 million recorded under comprehensive earnings, with reclassifications to interest expense planned over ten years.

Description	**2016 ($ in mill.)**	**2015 ($ in mill.)**
Billed Trade Receivables	1452	1546
Unbilled Trade Receivables	228	201
Total Trade Receivables	1680	1747
Allowance for Doubtful Accounts	-41	-16
Net Trade Receivables	1639	1731

Question: what was the percentage change in total trade receivables net from 2015 to 2016?

Answer: -5%

A.3 Model's Generated Response

A.3.1 Fine-tuned Llama-3-8b Model

A.3.1.1 Example 1

Context: AMERICAN TOWER CORPORATION AND SUBSIDIARIES NOTES TO CONSOLIDATED FINANCIAL STATEMENTS (Tabular amou ...
Question: What were note receivables in 2019?
Generated output: 1.1 million
Reference: 1.1
Permissive Accuracy: 1
Exact Match Accuracy: 1

A.3.1.2 Example 2

Context: SUMMARY STATEMENTS OF FINANCIAL POSITION Note: (1) 'Currency translation reserve' relates mainly to ...
Question: What is the average share capital across the 2 years?
Generated output: 4275.5
Reference: 4127
Permissive Accuracy: 0
Exact Match Accuracy: 0

A.3.1.3 Example 3

Context: In December 2014, investors who had purchased American Depository Shares (ADSs) of Petrobras on the ...
Question: Given the input sentence, please extract the subject and object containing a certain relation in the sentence according to the following relation types, in the format of "relation1: word1, word2; relation2: word3, word4". Relations include: product/material produced; manufacturer; distributed by; industry; position held; original broadcaster; owned by; founded by; distribution format; headquarters location; stock exchange; currency; parent organization; chief executive officer; director/manager; owner of; operator; member of; employer; chairperson; platform; subsidiary; legal form; publisher; developer; brand; business division; location of formation; creator.
Generated output: stock_exchange: Petrobras, New York Stock Exchange; headquarters_location: Petrobras, New York City; location_of_formation: Petrobras, Brazil; cur-

rency: Petrobras, Brazilian real; stock_exchange: Petrobras, Brazilian Stock Exchange; headquarters_location: Petrobras, Rio de Janeiro; location_of_formation: Petrobras, Brazil; currency: Petrobras, Brazilian real; stock_exchange: Petrobras, New York Stock Exchange; headquarters_location: Petrobras, Rio de Janeiro; location_of
Reference: stock_exchange: Petrobras, New York Stock Exchange
Permissive Accuracy: 1
Exact Match Accuracy: 0

A.3.2 Llama-3-8b Model with FSP+CoT

Note: This example demonstrates the Few-Shot Prompting (FSP) with Chain of Thought (CoT) approach. The context and step-by-step reasoning are shown to illustrate how the model processes information and arrives at an answer.

A.3.2.1 Example: 1

Generated Output: You are a financial expert assistant. Answer the following question based on the given context. Use step-by-step reasoning and provide numerical calculations where applicable. If asked to extract subject and object in relation, return only the single most relevant and applicable extraction. Use ' ' for line breaks in your response.

Examples: Example 1: [Truncated for brevity] entergy texas, inc. and subsidiaries management 2019s financial discussion and analysis results of operations ... Table: amount (in millions) 2016 net revenue $ 644.2 net wholesale revenue -35.1 (35.1) purchased power capacity -5.9 (5.9) transmission revenue -5.4 (5.4) reserve equalization 5.6 retail electric price 19.0 other 4.4 2017 net revenue $ 626.8

what percent did net revenue decrease between 2016 and 2017?

Answer: Calculation: 1. 2016 net revenue: $644.2 million 2. 2017 net revenue: $626.8 million 3. Decrease = $644.2 million - $626.8 million = $17.4 million 4. Percent decrease = ($17.4 million / $644.2 million) * 100 = 2.7% Answer: 2.7% Now, please answer the following:
Actual Context: SUMMARY STATEMENTS OF FINANCIAL POSITION Note: (1) 'Currency translation reserve' relates mainly ...
Actual Question: What is the average share capital across the 2 years?

Answer (including calculation steps and final answer, or single most relevant relation extraction, use ' ' for line breaks): 4127.5
Reference: 4127

Permissive Accuracy: 0
Exact Match Accuracy: 0

A.3.2.2 Example 2

Context: AMERICAN TOWER CORPORATION AND SUBSIDIARIES NOTES TO CONSOLIDATED FINANCIAL STATEMENTS (Tabular amou ...
Question: What were note receivables in 2019?
Generated output: 1.1
Reference: 1.1
Permissive Accuracy: 1
Exact Match Accuracy: 1

A.3.2.3 Example 3

Context: Slide 25 demonstrates how our Banquet, Healthy Choice and Marie Callender's brands are leading froze ...
Question: Given phrases that describe the relationship between two words/phrases as options, extract the word/phrase pair and the corresponding lexical relationship between them from the input text. The output format should be "relation1: word1, word2; relation2: word3, word4". Options: product/material produced, manufacturer, distributed by, industry, position held, original broadcaster, owned by, founded by, distribution format, headquarters location, stock exchange, currency, parent organization, chief executive officer, director/manager, owner of, operator, member of, employer, chairperson, platform, subsidiary, legal form, publisher, developer, brand, business division, location of formation, creator.
Generated output: industry: Marie Callender's, food; product_or_material_produced: Marie Callender's, frozen food; owned_by: Marie Callender's, Conagra Brands; parent_organization: Marie Callender's, Conagra Brands; subsidiary: Conagra Brands, Marie Callender's; owned_by: Banquet, Conagra Brands; parent_organization: Banquet, Conagra Brands; subsidiary: Conagra Brands, Banquet; owned_by: Healthy Choice, Conagra Brands; parent_organization:
Reference: distribution_format: frozen, single
Permissive Accuracy: 0
Exact Match Accuracy: 0

A.3.3 Tool-based Sample

Context: Cloud & Cognitive Software * Recast to reflect segment changes. ** 2019 results were impacted by Red...

Question: What percentage of Cloud & Cognitive Software external revenue was Transaction Processing Platforms in 2019?

Code:

```
transaction_processing_platforms_revenue_2019 = 7936
cloud_cognitive_software_revenue_2019 = 23200
percentage_transaction_processing_platforms_2019 = (
    transaction_processing_platforms_revenue_2019 /
    cloud_cognitive_software_revenue_2019
) * 100
print(f'Percentage of Cloud & Cognitive Software external revenue '
      f'that was Transaction Processing Platforms in 2019: '
      f'{percentage_transaction_processing_platforms_2019:.2f}%')
```

Prediction: Percentage of Cloud & Cognitive Software external revenue that was Transaction Processing Platforms in 2019: 34.21%

Reference: 34.21

Exact Match Accuracy: 1

A.3.4 Llama-3-8b-instruct (Code gen)

Context: Operational performance A Leasing activity We agreed 205 long-term leases in 2019, amounting to £26 million annual rent ...

Question: What is the percentage change in the number of opened or refitted unites in the centres from 2018 to 2019?

Generated Code:

```
{
    "number_of_opened_units_2018 = 262\n
    number_of_opened_units_2019 = 256\n
    difference_in_opened_units = abs(number_of_opened_units_2019 -
    number_of_opened_units_2018)\n
    percentage_change = (difference_in_opened_units /
    number_of_opened_units_2018) * 100\n
    print(percentage_change)"
```

}

Final answer after execution: -2.2900763358778624

Reference: -2.29

Permissive Accuracy: 1

Exact Match Accuracy: 0

A.4 Plots and Metrics

1. Bar Plot for Code Gen

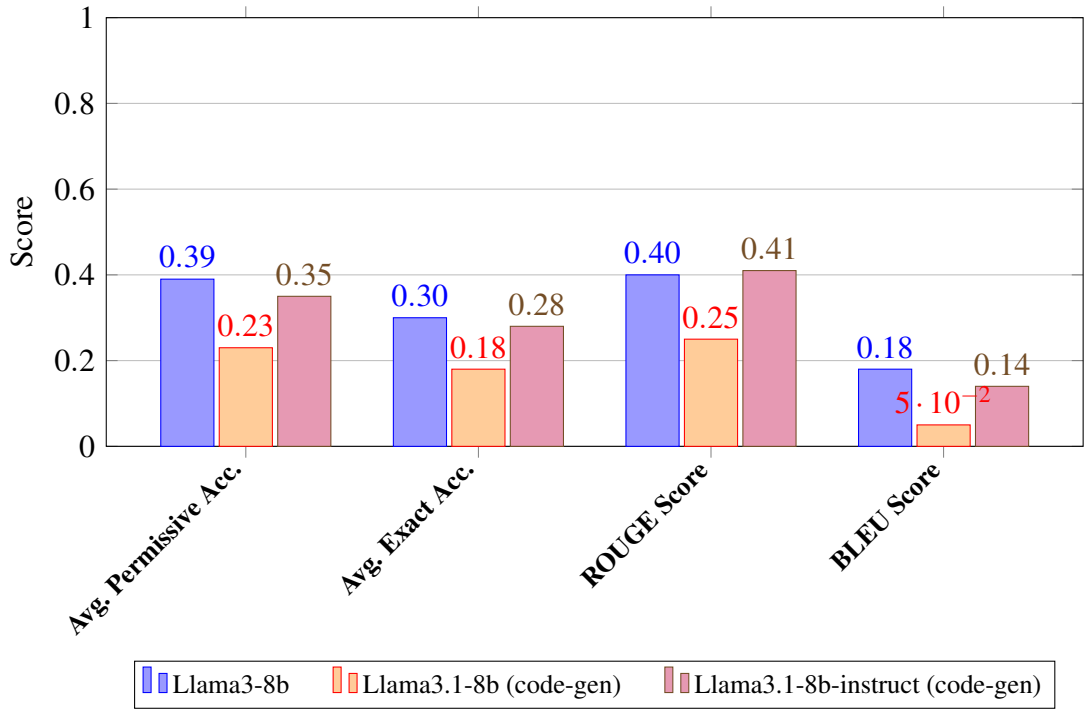

Figure A.1: Evaluation Metrics for Code Generation compared with the base model

2. Training Metrics

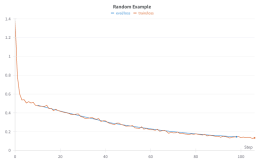

Figure A.2: Training Curve for One Random Example

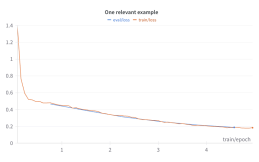

Figure A.3: Training Curve for One Relevant Example

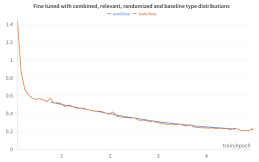

Figure A.4: Training Curve for Mixed Type Examples