Large language model number handling in the Finance Domain

Anant Raj

Master of Science

School of Informatics
University of Edinburgh

2024

Abstract

This research project focuses on enhancing the performance of large language models (LLMs) in finance by improving their capabilities in numerical data handling and financial reasoning. By employing a mixture of financial datasets our approach encompasses specialized instruction tuning methods alongside the strategic use of specialised external computational tools for specific tasks. The project rigorously fine-tuned open-source LLMs like Llama2,Llama3 and evaluate their performance in comparison with commercial models such as GPT-4. The principal aim is to forge a robust framework for the processing and analysis of financial data, which minimizes human intervention while enhancing accuracy and efficiency, thus redefining the benchmark for AI applications in financial analysis.

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

I am profoundly grateful to my supervisor, Alexandra Birch, whose expertise, understanding, and patience added substantially to my graduate experience. I deeply appreciate the countless hours she spent discussing experimental results with me and her insightful comments that continually inspired me throughout this journey. Her comprehensive guidance and unwavering support were instrumental in the successful completion of this project.

I extend my heartfelt thanks to Mateusz, whose encouragement was a beacon of light during times of struggle. His motivating words helped me navigate through the challenging moments of my research.

My sincere gratitude also goes out to my friends and family, whose unending support and love provided me with strength and encouragement. Their belief in my abilities sustained me throughout this academic endeavor.

This dissertation stands as a testament not only to my efforts but also to the valuable contributions of all those who supported me along the way. Thank you.

Table of Contents

Chapter 1

Introduction

The intersection of Large Language Models (LLMs) and the financial industry represents a frontier of considerable promise and complexity. As documented by recent advancements, LLMs such as ChatGPT and GPT-4 have showcased remarkable proficiencies across a spectrum of Natural Language Processing (NLP) tasks, driven by sophisticated training methodologies including reinforcement learning from human feedback (RLHF) and masked language model objectives [15]. However, recent advancements have highlighted their limitations, such as the inability to access current information, difficulties with arithmetic and mathematical computations, and a propensity to generate inaccurate responses due to hallucinations [22, 13]. Despite being trained on diverse datasets covering multiple genres and subjects, their effectiveness in specialized domains like finance still necessitates further scrutiny.

In the financial arena, LLMs have begun to play an indispensable role, aiding in tasks such as investment sentiment analysis, financial named entity recognition, question-answering systems, and stock market prediction, which assist financial analysts in navigating complex datasets and predictive models. Methods such as multi-field LLMs and instruction fine-tuned LLMs have been explored to enhance the efficacy of generated results in the financial context [11]. Nonetheless, the nuanced understanding and processing of financial data by LLMs — critical for applications ranging from market analysis to investment strategy formulation — remain at an early stage for empirical study and innovation.

This project aims to investigate various number-handling strategies to fine-tune a large language model using instruction data, with the goal of achieving optimal performance in finance-related downstream tasks, particularly those involving currency or numerical questions. A diverse set of financial datasets has been selected, encompassing tabular data, contextual information, relation extraction, and Conversational Finance Question Answering. These complex numerical reasoning datasets provide a robust foundation for generalizing across a wide range of financial tasks.

Initially, baseline performance was assessed on open-source models such as Llama2 and the newly launched Llama3 using these datasets. It is well known that large language models (LLMs) often struggle with numerical data, particularly in tasks requiring numerical reasoning, and our results were consistent with this observation. Several fine-tuning approaches were explored, starting with the Llama2-chat 7B model and subsequently extending experiments to the latest Llama3 8B model. Recognizing that a single prompting technique does not suffice for all scenarios, various techniques were experimented with, yielding notable results detailed in the experiment section.

In our study, we venture beyond the traditional confines of language model applications, drawing upon the groundbreaking work of Schick et al. (2023) who utilized external tools to elevate the performance of models handling numerical values over those processing textual data. This approach aligns with cutting-edge developments like LangChain agents and function calling, which empower models to leverage external capabilities effectively.

Our experiments are not merely replications but expansions, tailored to explore uncharted territories in financial analytics. We innovated upon existing methodologies by implementing a sophisticated pipeline structure. Initially, financial datasets serve as input for code-generating large language models (LLMs), tasked with crafting precise Python scripts based on the nuanced demands of the financial context. These scripts are then executed by another model functioning within a Python REPL environment, aimed at delivering the final outputs.

Given the inherent challenges of computational limitations and the absence of open-source LLMs capable of processing extensive contextual inputs typical of financial datasets, we proceeded under the hypothesis that, with optimal resources and fine-tuning, our language model could refine its script generation to achieve unprecedented accuracy levels, potentially surpassing current benchmarks. Our pilot study, utilizing a compact dataset sample with the 'facebook/opt-1.3b' model in conjunction with LangChain's Python agent, python_repl, yielded promising results. These initial findings not only demonstrate the model's capability but also mark a pivotal first step towards scaling up and enhancing the precision and analytical prowess of LLMs when faced with larger and more complex data sets. This sets a robust foundation for future research to build upon, aiming for significant advancements in the application of LLMs to financial tasks.

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 Large Language Models, outlining key milestones and innovations. Section 2.3 examines the advancements in LLM capabilities, highlighting their impact on the field of NLP.

2.1 The Transformer Architecture: A Paradigm Shift

The introduction of the Transformer architecture by Vaswani et al. (2017) marked a pivotal moment in NLP, fundamentally changing the approach to sequence transduction problems. Unlike previous recurrent or convolutional neural networks, the Transformer relies solely on attention mechanisms, enabling more efficient processing of sequential data and facilitating the training of much larger models [23].

Key components of the Transformer architecture include:

- Self-Attention Mechanism This allows the model to weigh the importance of different parts of the input sequence when processing each element, enabling capture of long-range dependencies more effectively than previous architectures.
- Multi-Head Attention By applying multiple attention operations in parallel, the model can capture different types of relationships within the data simultaneously.

- Positional Encoding: To compensate for the lack of inherent sequential processing, positional encodings are added to input embeddings, allowing the model to leverage sequence order.
- Feed-Forward Networks These process the outputs of the attention layers, adding non-linearity and increasing the model's capacity to learn complex functions.
- Layer Normalization and Residual Connections These components facilitate training of deep networks by stabilizing the learning process and mitigating the vanishing gradient problem.

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 of Large Language Models

Building upon the Transformer architecture, researchers developed a series of increasingly powerful language models:

BERT: Bidirectional Encoders

BERT (Bidirectional Encoder Representations from Transformers), introduced by Devlin et al. (2018), represented a significant advancement in pre-training techniques [7].

Key innovations of BERT include:

- Bidirectional Context: Unlike previous models that processed text either left-to-right or right-to-left, BERT considers both left and right context simultaneously, enabling a more nuanced understanding of language.
- Masked Language Model (MLM) Pre-training: BERT is trained to predict masked words in a sentence, forcing it to learn contextual representations of words.
- Next Sentence Prediction: This additional pre-training task helps BERT understand relationships between sentences.

BERT's approach led to state-of-the-art performance across a wide range of NLP tasks, demonstrating the power of large-scale, unsupervised pre-training.

GPT Series: Scaling Up

The GPT (Generative Pre-trained Transformer) series, developed by OpenAI, pushed the boundaries of model size and capabilities:

- GPT : Introduced the concept of fine-tuning a large, pre-trained language model for specific downstream tasks [19].
- GPT-2 : Scaled up the model size significantly (1.5 billion parameters) and demonstrated impressive text generation capabilities [20].
- GPT-3 : With 175 billion parameters, GPT-3 marked a leap in scale and showcased strong few-shot learning abilities, reducing the need for task-specific fine-tuning [3].

The GPT series highlighted the benefits of scale in language models, showing that larger models could exhibit more flexible and generalizable language understanding and generation capabilities.

Other Notable Models

- T5 : Unified various NLP tasks into a single text-to-text format, demonstrating the versatility of the transformer architecture [21].
- BART : Combined the bidirectional encoder of BERT with the autoregressive decoder of GPT, showing strong performance on both text comprehension and generation tasks [12].

These models further expanded the capabilities of LLMs, showing their potential to handle a diverse range of NLP tasks within a single architectural framework.

2.3 Advancements in LLM Capabilities

As LLMs evolved, several key advancements emerged:

Scaling Laws

Kaplan et al. (2020) established important relationships between model size, dataset size, and computational budget [10]. Their findings showed that model performance scales predictably with these factors, guiding the development of more efficient and powerful models. This work provided a theoretical foundation for the ”bigger is better” approach in language model development.

Few-shot and Zero-shot Learning

Brown et al. (2020) demonstrated with GPT-3 that sufficiently large models could perform tasks with minimal or no task-specific training examples [3]. This capability, known as few-shot and zero-shot learning, opened new possibilities for creating more flexible and adaptable AI systems.

Instruction Tuning

Wei et al. (2022) showed that fine-tuning models on diverse sets of instructions could significantly improve their ability to follow natural language prompts [26]. This technique, known as instruction tuning, enhanced the versatility of LLMs across various tasks and made them more aligned with human intentions.

Chain-of-Thought Prompting

Another significant advancement came from Wei et al. (2022), who demonstrated that prompting models to generate step-by-step reasoning could substantially improve their performance on complex tasks, including those involving numerical reasoning [27]. This technique, called chain-of-thought prompting, showed that LLMs could be guided to break down complex problems into more manageable steps, mirroring human problem-solving processes.

Reinforcement Learning from Human Feedback (RLHF)

Ouyang et al. (2022) introduced techniques to align language models with human preferences using reinforcement learning [18]. This approach, known as RLHF, led to the development of models that could produce more helpful, safe, and contextually appropriate outputs.

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 [16]. 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 [28]. 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 [24]. 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 [5]. 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 [25]. 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 [1]. 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 [29]. 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 of fine-tuning large models for specific tasks, Dettmers et al. (2023) proposed QLoRA (Quantized Low-Rank Adaptation), a parameter-efficient fine-tuning method [6]. Their approach involves quantizing the pretrained language model to 4 bits, adding trainable low-rank adapters, and using a novel preconditioner which they term ”Paged Optimizers” to handle GPU memory constraints. This method could potentially allow for more efficient adaptation of large, general-purpose LLMs to specific financial tasks without the need for extensive computational resources. 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 [22]. 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 more so in a complex domains like finance.

Chapter 4

Methodology

This chapter describes the methodology applied in this project.

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 the appendix section.

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 [30].

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 [5].

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 [4].

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

- Algorithm Development: A preprocessing algorithm 1 was created to prepare the merged dataset for model training. This algorithm ensures that the data follows a unified structure comprising context, questions, and answers.

- Prompt Template: The data was formatted into a specific template prompt format required by both Llama2 7b and Llama3 8b models that are used in this project. This involved:
  - Data Cleaning: Removing inconsistencies and irrelevant information.
  - Normalization: Standardizing numerical values and textual content to maintain uniformity.
  - Prompt Structuring: Organizing the data into the structured prompts that the models were originally trained on, ensuring compatibility and effectiveness during fine-tuning. This include use of special tokens and prompt structure of the specific model [2].

4.2 Baseline Model Selection and Evaluation

- Baseline Models Used:
  - meta-llama/Llama-2-7b-hf [3]

[1] https://huggingface.co/datasets/FinGPT/fingpt-finred-re
[2] https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3
[3] https://huggingface.co/meta-llama/Llama-2-7b-hf

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

  - meta-llama/Meta-Llama-3-8B [4]

- Performance Testing: The merged dataset was tested on these baseline models to establish initial performance metrics. This step was crucial for:
  - Benchmarking: Establishing a reference point for comparing improvements post fine-tuning.
  - Identifying Weaknesses: Understanding the initial strengths and weaknesses of the models with raw financial data.

4.3 Fine-tuning Approaches

4.3.1 Training Procedure

Model Architecture

We employed the Meta-Llama-2-7B-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 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 X experiment, we exclusively used the TAT-QA dataset, following a preprocessing strategy similar to the one outlined in the algorithm 1, specifically steps 20-25.

[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 [8]. 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 [8]. 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 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 [14]. 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 [17]. 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 our X (or last) experiment, conducted on a Google Colab environment with a single NVIDIA A100 40GB GPU, we utilized the newly released model META-LLAMA/META-LLAMA-3.1-8B model to generate Python code for financial analysis, employing a novel approach without fine-tuning the model.

Evaluation Strategy

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

Few-Shot Example Configuration

In the experiments 5.? section we have mentioned the results using various few shot learning techniques. Few-shot learning (FSL) involves training models to recognize

[6] https://huggingface.co/docs/accelerate/en/usage$_{g}uides/gradient_{a}ccumulation$

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 [2]. 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 [9]. 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 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," 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 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 Experiment -X, 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)
    inputs ← embedding_tokenizer(text, return_tensors='pt', padding=True, truncation=True)
    embedding ← embedding_model(**inputs).last_hidden_state.mean(dim=1).cpu().numpy()
    return embedding[0]
end function
function PRECOMPUTERELEVANTEXAMPLES(dataset, few_shot_examples, top_k, context_weight, question_weight)
    few_shot_context_embeddings ← [GetEmbeddings(ex["context"]) for ex in few_shot_examples]
    few_shot_question_embeddings ← [GetEmbeddings(ex["question"]) for ex in few_shot_examples]
    relevant_examples_map ← {}
    for idx, item in enumerate(dataset) do
        context_embedding ← GetEmbeddings(item['context'])
        question_embedding ← GetEmbeddings(item['question'])
        context_similarities ← cosine_similarity([context_embedding], few_shot_context_embeddings)[0]
        question_similarities ← cosine_similarity([question_embedding], few_shot_question_embeddings)[0]
        combined_similarities ← context_weight * context_similarities + question_weight * question_similarities
        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 available in my Github Repo[7].

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 that accommodates minor precision differences by allowing slight deviations in decimal points within an alpha threshold. This measure is particularly useful in financial contexts, where exact numerical precision

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

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 Sequence-Matcher is used for string comparison, returning 1 if the similarity exceeds 0.9 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 Calculate Accuracy with Some Leniency

```
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) < 0.001)
    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 > 0.9 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 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: Measures overlap of n-grams between generated and reference texts.
- METEOR: Considers precision, recall, and synonymy for evaluating translation quality.
- BLEU Score: Evaluates the precision of n-grams in generated text against reference.
- BERT Precision, Recall, F1 Score: Uses contextual embeddings to measure semantic similarity.
- GLUE Benchmark: It is a collection of resources for training, evaluating, and analyzing natural language understanding systems across diverse tasks.

4.5 Few-Shot Code Generation & External Tool Integration

4.5.1 Subset Selection and Preparation

First, a small subset of the dataset was curated specifically for the LangChain integration. This subset was chosen to represent a variety of financial question-answer scenarios that required detailed computational analysis. Each entry in this subset included Python scripts in the answer fields, which were necessary for executing complex calculations. The selection criteria ensured that the subset contained diverse and representative examples that could test the model's ability to handle different types of financial queries.

4.5.2 Python REPL Pipeline Integration

To execute the Python scripts included in the answer fields, the subset was run through a Python REPL (Read-Eval-Print Loop) pipeline. The Python REPL pipeline allows for real-time execution of Python code, which is crucial for dynamic and accurate computation of financial data. The model utilized for this integration was the Facebook 1.5b model, known for its robust language processing capabilities.

4.5.3 Integration Process

The integration process involved several technical steps:

- Embedding Python Scripts: The Python scripts were embedded directly into the model prompts. This required careful formatting to ensure the scripts were correctly interpreted and executed by the REPL pipeline.

- Model Adaptation: The Facebook 1.5b model was adapted to recognize and handle the embedded Python scripts. This adaptation involved training the model to parse the script correctly and execute it within the context of the financial question-answering task.

- Output Handling: The outputs generated by the Python scripts were then reintegrated into the model's response. This involved capturing the output from the REPL pipeline, formatting it appropriately, and ensuring it was included in the final answer provided by the model.

4.5.4 Execution and Validation

During the inference phase, the integrated system worked as follows:

- Query Processing: When a financial query was posed, the model parsed the question and identified the need for computational analysis.

- Script Execution: The relevant Python script was executed using the Python REPL pipeline. This real-time execution allowed the model to handle complex calculations dynamically.

- Result Integration: The results from the script execution were captured and formatted to be part of the model's response. This integration ensured that the answer provided was accurate and directly addressed the computational aspects of the query.

Chapter 5

Experiments

5.1 Comparative Analysis

5.2 Fine-tuning Strategy Optimization

5.3 Cross-task Generalization

5.4 Results and Discussion

Chapter 6

Conclusions

6.1 Limitations

6.2 Future Work

Bibliography

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

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

[9] 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.

[10] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

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

[12] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

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

[19] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[22] 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.

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

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

Appendix A

First appendix

A.1 Examples

Below are some sample examples comparing the model's predictions to the reference values. The best performing model, highlighted in the table earlier, is used to generate these predictions.