Fine-tuning Smaller Language Models for Question Answering over Financial Documents

Karmvir Singh Phogat, Sai Akhil Puranam, Sridhar Dasaratha,
Chetan Harsha, Shashishekar Ramakrishna
EY Global Delivery Services India LLP
{Karmvir.Phogat,Sai.Puranam,Sridhar.Dasaratha}@gds.ey.com,
{Chetan.Harsha,Shashishekar.R}@gds.ey.com

Abstract

Recent research has shown that smaller language models can acquire substantial reasoning abilities when fine-tuned with reasoning exemplars crafted by a significantly larger teacher model. We explore this paradigm for the financial domain, focusing on the challenge of answering questions that require multi-hop numerical reasoning over financial texts. We assess the performance of several smaller models that have been fine-tuned to generate programs that encode the required financial reasoning and calculations. Our findings demonstrate that these fine-tuned smaller models approach the performance of the teacher model.

To provide a granular analysis of model performance, we propose an approach to investigate the specific student model capabilities that are enhanced by fine-tuning. Our empirical analysis indicates that fine-tuning refines the student models ability to express and apply the required financial concepts along with adapting the entity extraction for the specific data format. In addition, we hypothesize and demonstrate that comparable financial reasoning capability can be induced using relatively smaller datasets.

1 Introduction

In recent years, the development of large language models (LLMs) has achieved significant advances in natural language processing (NLP). Scaling up the size of these models has not only improved sampling efficiency and performance, (Kaplan et al., 2020) but also introduced reasoning capabilities (Wei et al., 2022b; Kojima et al., 2022). LLMs have been shown to perform well on tasks requiring reasoning capabilities in various domains, including code writing (Chen et al., 2021a), math problem solving (Lewkowycz et al., 2022; Polu et al., 2023), dialogue (Glaese et al., 2022; Thoppilan et al., 2022), common sense reasoning (Shwartz et al., 2020; Chowdhery et al., 2022) and symbolic reasoning (Wei et al., 2022a; Wang et al., 2023b).

A major drawback of these methods is their reliance on extremely large models with hundreds of billions of parameters. These models can be costly to deploy at scale due to high computational requirements and inference costs. To overcome these limitations, recent research has focused on inducing reasoning in smaller models. A common approach is to use large models to generate training samples with demonstrations of reasoning on one or more tasks that are then used to fine tune smaller models (Magister et al., 2023; Ho et al., 2023). While these methods have shown promising results on various tasks including arithmetic, symbolic and common-sense reasoning, the applicability, and effectiveness of these methods in specific domains such as finance need further exploration. Question answering in the finance domain poses a unique set of challenges, requiring the understanding of financial concepts along with the ability to perform numerical reasoning. This complexity introduces a significant challenge that is distinct from classical question answering problems (Yang et al., 2018; Rajpurkar et al., 2018)

In this paper, we present an empirical study that provides experimental evidence supporting the effectiveness of fine-tuning small language models for financial question answering. Our research is guided by several critical questions:

RQ1: To what degree does fine-tuning small language models improve their performance on financial question answering tasks?

RQ2: What are the intrinsic characteristics of the base language model that contribute to its performance prior to fine-tuning?

RQ3: Which aspects of question answering benefit directly from the pre-trained knowledge, and what specific improvements are enabled by fine-tuning?

RQ4: What are the fine-tuning data requirements to achieve these improvements?

To address these questions, we adapt previous ap-

Findings of the Association for Computational Linguistics: EACL 2024, pages 10528–10548
November 12-16, 2024 ©2024 Association for Computational Linguistics

proaches on inducing reasoning in smaller models for the financial question answering task. In our experimental setup, we employ GPT-4 as the teacher model, building upon its documented success in the realm of financial question answering (Chen et al., 2023; Phogat et al., 2023). For the student models, we explore a suite of state-of-the-art, yet relatively smaller, language models including PHI-3 variants (3.5B and 14B parameters) (Abdin et al., 2024), MISTRAL 7B, and ORCA-2 configurations (7B and 13B parameters) (Mitra et al., 2023). Our methodology involves training the student model using Python programs generated by the teacher model. The teacher generated code systematically delineates the steps for financial reasoning, including question comprehension, formula identification and entity extraction. At inference time, the language models are tasked with producing Python code, which is then executed by an external interpreter. This code generation and external execution strategy has been demonstrated to be more effective than methods that rely on the language model to internally perform the calculations. (Gao et al., 2023; Chen et al., 2023; Phogat et al., 2023).

Our main contributions are summarized as follows:

1. We refine the process of fine-tuning smaller language models by introducing an approach designed for question answering over financial reports.

2. Our experimental study on three financial datasets provides insights on the performance of the small models

3. We propose an evaluation method that utilizes GPT-4 to assess the Python code outputs from the student models, pre- and post-fine-tuning. Combining GPT-4's automated assessment with manual evaluation yields new insights into the distinct competencies that are enhanced in the student model during fine-tuning.

4. Motivated by these insights we explore the use of smaller datasets for fine-tuning and provide empirical evidence demonstrating their effectiveness.

Our experimental findings reveal that smaller language models fine-tuned for financial reasoning, can achieve performance that rivals that of the larger teacher model. Moreover, our empirical analysis suggests that the fine-tuning helps refine concept understanding and enables consistent reasoning with those concepts. In addition, the entity extraction abilities get honed on the specific format of the financial dataset.

2 Background

Pre-trained large language models are shown to perform well on tasks requiring reasoning when used with certain techniques. (Wei et al., 2022b) proposed prompting the LLMs to solve the problem step-by-step by providing a few exemplars. (Chen et al., 2023) proposed a few-shot prompt to produce a program which is then executed externally. However, methods relying on pre-trained LLMs can be costly to deploy at scale.

Recent efforts attempt to replicate these reasoning capabilities in small language models. One of the common approaches is following a teacher-student setup where a pre-trained LLM acts as a teacher, generating training data which is used to teach a small language model, the student. (Mukherjee et al., 2023; Mitra et al., 2023) aim to train models to exhibit generic reasoning abilities. They utilize GPT-3.5 TURBO and GPT-4 as teacher models to generate training data with carefully crafted prompts. On the other hand, (Fu et al., 2023; Magister et al., 2023; Ho et al., 2023) train task specific small language models with CoT based explanation from pre-trained LLMs. Specifically for problems involving mathematical reasoning, (Wang et al., 2023a; Gou et al., 2023; Toshniwal et al., 2024; Wang et al., 2023c) propose to generate programs from the pre-trained LLMs and train the small language models. In contrast, we focus on fine-tuning small language models for financial question answering problems (Chen et al., 2021b, 2022; Zhu et al., 2021). Solving these problems requires financial domain knowledge and complex reasoning compared to the math word problems addressed in previous studies.

Prior works have studied the use of pre-trained LLMs for financial question answering. (Srivastava et al., 2024) perform a detailed comparison of the performance of pre-trained LLMs on financial datasets with various prompting techniques. They also introduce a novel prompting technique optimized for semi-structured documents. (Phogat et al., 2023) developed zero-shot prompt templates for question answering over financial documents that guide the LLMs to encode reasoning into a python program. These efforts do not consider fine-

tuning to specialize language models for the financial domain. (Theuma and Shareghi, 2024) explore task-wise integration of external tools, calculator and SQL engine, with fine-tuned small language models for tabular data analysis in finance. However, their approach utilizes predetermined prompt templates for data generation rather than a teacher-student approach for fine-tuning. In our approach, we generate structured python programs using pre-trained LLMs and train small language models with supervised fine-tuning. Furthermore, we examine in detail the nature of the alignment achieved by the fine-tuning for the financial question answering, an area that is largely unexplored in previous research.

3 Fine-tuning for Financial Question Answering

We adopt the approach of using very large-scale models as reasoning teachers, and fine-tuning relatively small-scale student models from the prompt-completion pairs generated using the teacher model (Hsieh et al., 2023; Ho et al., 2023). Specifically, the large model is used to generate python code with comments that encapsulates the reasoning required to answer the financial question. Furthermore, the programs are generated with a specific structure that facilitates subsequent performance analysis of the fine-tuned model, as discussed in detail in Sec. 5. The fine-tuning task is performed in three steps as shown in Figure 1.

3.1 Code Generation

In the code generation step, we employ the teacher model to generate a Python code for a specified question-answering task. Financial question answering consists of three distinct steps: understanding the concept and writing the formula required to answer the question, finding the relevant entities, and then executing and storing the calculations. A sample of the desired code structure encapsulating this reasoning process is show in in Figure 2. We guide the teacher model to consistently generate the desired code structure through program of thought (PoT) prompting (Chen et al., 2023). In few-shot PoT prompting, as shown in Figure 1, few shot exemplars are prefixed as demonstrations for the teacher model to generate codes in the desired format.

In the financial question answering training dataset, each sample contains the final answer to the question and additionally it may contain a program which demonstrate step-wise arithmetic calculations that are required to be performed to arrive at the answer. We incorporate these programs as answer hints in the few-shot PoT prompt to guide the teacher model towards accurate code generation. This strategy can potentially improve the question-answering accuracy of the teacher model.

3.2 Data Curation

During data curation, we filter out the samples with incorrect teacher codes and format the filtered samples to prompt-completion pairs for the student model. At the filtering stage, the samples with incorrect teacher-generated codes are identified by executing the codes and comparing the resulting answer with the ground truth answer. The filtered samples are then formatted to prompt-completion pairs as per student model requirements. For instance, the prompt instructions for the MISTRAL 7B (instruction tuned) model should begin with the token [INST] and ends with the token [/INST]. In addition, special characters like '#' can be used to symbolize the prompt structure.

3.3 Fine-tuning

The fine-tuning task for question answering is represented by the prompt-completion pairs: $\mathcal{D} = \{x_i, y_i\}_{i=1}^{N}$ where x_i is a token sequence for the fine-tuning prompt and y_i is a token sequence for the corresponding code completion, as shown in Figure 1. We use low rank adaptation (LoRA) (Hu et al., 2022), a special class of parameter efficient fine-tuning that takes advantage of the low "intrinsic dimension" of pre-trained LLMs, when adapting to a specific task (Aghajanyan et al., 2021). The student model's LoRA adapter is fine-tuned to adapt to the financial question answering task.

4 Experiments

4.1 Experimental Design

Datasets: We conduct our experiments on three English language financial question answering datasets FinQA (Chen et al., 2021b), ConvFinQA (Chen et al., 2022) and TATQA (Zhu et al., 2021). The question answering task, in FinQA and TATQA datasets, is to answer the questions using the passage containing text and table content. The experiments for TATQA are restricted to questions of *arithmetic* type. In ConvFinQA, the task is to answer the last question from a conversation containing a series of interrelated questions, based on

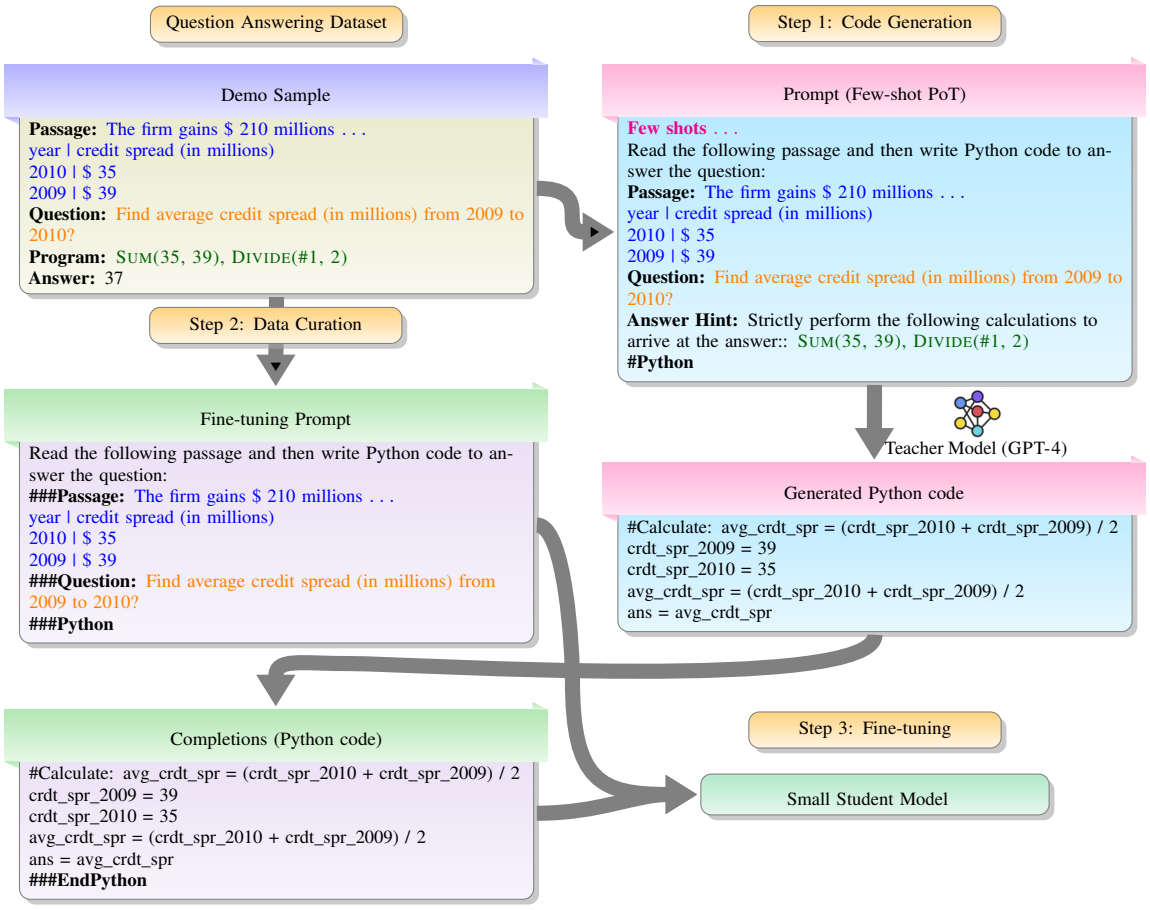

Figure 1: Fine-tuning for financial question answering.

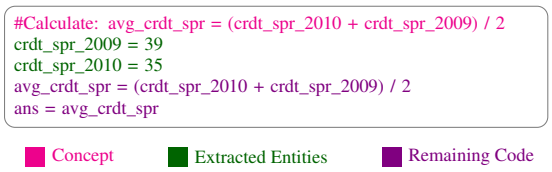

Figure 2: GPT-4 generated sample Python code

a given text and table content. The table content is represented in a textual format using the strategy adopted in (Chen, 2022).

Fine-tuning: In our experiments, for the teacher model we use the GPT-4 model provided by Azure OpenAI[1]. For student models, we consider the following open source family of models: PHI-3, MISTRAL and ORCA-2. We provide further details on the models in Table 7 in the Appendix. In the code generation step, we use GPT-4 as a teacher, and prompt it with 4-shot exemplars for all datasets. These exemplars are derived from few-shot PoT prompts as discussed in (Chen et al., 2023). The few-shot teacher's prompt for FinQA, ConvFinQA and TATQA dataset is given in Figure 18, Figure 19 and Figure 20 in the Appendix E respectively. In each of the datasets, an expression encoding the required calculations, is provided for each training sample. We include the expression as a hint in the prompt for the GPT-4 model. In the data curation step, any data that contains incorrect GPT-4 code is eliminated. The filtered data is then converted into prompt-completion pairs to meet the requirements of the student model. Finally, during the fine-tuning stage, the student model is trained using these prompt-completion pairs. For fine-tuning the student models, we use LoRA (Hu et al., 2022) method of HuggingFace PEFT library[2] that back-propagates gradients to low rank adapters.

In our fine-tuning experiments, the datasets are divided into train, dev, and test splits. The FinQA dataset includes predefined splits with their ground truths. As the test splits for ConvFinQA and TATQA datasets don't include ground truths, we use the predefined dev splits of these datasets as the test splits. The dev splits for ConvFinQA and

[1]https://oai.azure.com/

[2]https://huggingface.co/docs/peft/en/index

TATQA are created by randomly picking 10% samples of their predefined train splits and the remaining samples are used for the training. The datasets with their splits are summarized in Table 1. The

Dataset	Train	Dev	Test
FinQA	6251	883	1147
ConvFinQA	2737	300	421
TATQA†	4992	550	718

† Only arithmetic questions are considered.

Table 1: Datasets for fine-tuning

train and dev splits are used for the student model's fine-tuning and validation respectively. We select the model and the hyperparameters[3] that give the highest performance on the dev split, and report the fine-tuned model's performance on the test split. We employ the vLLM[4] framework for conducting inference on fine-tuned models. The experiments are performed on a compute instance with 24 cores, 220GB RAM and a A100 GPU (80GB).

Evaluation Metrics: The fine-tuned LLMs are trained to generate Python codes that are executed using the Python `exec` function to determine the resulting answer. The resulting answer is then compared against the ground truth, using the method described in (Phogat et al., 2023).

Baselines: We evaluate the base version of the student models using zero-shot and few-shot prompting.

Zero-shot prompting: We performed prompt engineering experiments for synthesizing the zero-shot prompt that directs the LLMs to generate Python code to answer the question. For the FinQA dataset, the optimized zero-shot prompt for the PHI-3, ORCA-2 and MISTRAL models is given in Figure 3 in the Appendix.

Few-shot prompting: The few-shot prompt includes a few example demonstrations for LLMs to learn in-context. For the FinQA dataset, the few-shot prompt for PHI-3, ORCA-2 and MISTRAL models is given in Figure 4 in the Appendix.

4.2 Fine-tuning Results

The performance of the fine-tuned LLMs is reported in Table 2. For comparison, we report the performance of zero-shot and few-shot prompts using GPT-3.5 TURBO and GPT-4. These results are taken from (Phogat et al., 2023). The zero-shot prompt methods are defined as ZS-FinPYT (GPT-3.5 TURBO) and ZS-FinPYT (GPT-4) while the few-shot methods are defined as Few-shot PoT (GPT-3.5 TURBO) and Few-shot PoT (GPT-4). In the results discussed below, the comparison of the fine-tuned models is done against the best-performing prompting method for a given dataset and GPT model. For conciseness we refer to only the model's name instead of the full method.

FinQA dataset: The zero-shot and few-shot performance of the models is highly varying with some models having low zero-shot accuracy while other achieve excellent zero shot performance, with one of the models surpassing GPT-3.5 TURBO. For some models, the few-shot results show significant improvement as compared to their zero-shot performance while for others the few-shot prompting results in a lower accuracy. The fine-tuned models show large improvements over their respective zero-shot and few-shot results. Despite major differences in performance of the base models, the fine-tuned versions achieve a similar accuracy within a 6% range. The results thus demonstrate that the proposed fine-tuning approach was effective across models with a wide range of performance for the base models. For fine-tuned models, the model size has a small effect with the larger ORCA-2 model being 2% more accurate than the smaller one while PHI-3-MEDIUM has 4% higher accuracy as compared to PHI-3-MINI. All the fine-tuned models outperform GPT-3.5 TURBO by 4%-10%, with the PHI-3-MEDIUM achieving an accuracy within 1% of the GPT-4. Overall, the results indicate that relatively small language models can be fine-tuned to be competitive with much larger models, for the financial question answering task.

ConvFinQA dataset: The results broadly follow a similar pattern to the results of FinQA but there are some differences. The differences between the fine-tuned models are slightly pronounced as compared to the FinQA dataset. The models are within 10% of each other with the effect of model size being more significant. For both ORCA-2 and PHI-3, the larger models achieve ∼5% higher accuracy than their respective smaller variants.

TATQA dataset: The results for TATQA closely mirror those observed for the FinQA dataset with all the models achieving excellent performance, their accuracies falling within a 5% range. The fine-tuned PHI-3-MEDIUM model excels for this

[3]The set of explored and optimal hyperparameters are provided in Table 6 in the Appendix A.

[4]https://github.com/vllm-project/vllm

dataset, marginally surpassing GPT-4. Model size has minimal effect for ORCA-2 while for PHI-3 a small effect is observed, with PHI-3-MEDIUM achieving 3% higher accuracy than PHI-3-MINI.

5 Performance Analysis

We examine the evolution of model capabilities during fine-tuning, seeking to identify specific model enhancements. To this end, we define, and measure three key capabilities (1) concept understanding measured by the ability to correctly identify the required calculation (2) entity extraction measured by the ability to extract all required entities and (3) generation of executable code. Our method relies on comparing the output of the student models with that of the teacher generated codes. Therefore, we remove all samples with incorrect teacher generated codes from further analysis.

5.1 Concept accuracy

Due to significant model output variation, it is hard to define a simple metric to measure concept accuracy. Motivated by the promising results shown by others (Zheng et al., 2023) in using LLM for evaluation in challenging scenarios, we propose the use of GPT-4 to rate the concept understanding demonstrated in the model output, using a 5-point scale defined as follows: 1: no understanding 2: limited understanding 3: partial understanding 4: mostly demonstrates understanding 5: perfect understanding. After some prompt engineering using 25 random student code samples, we found this method to provide reasonable assessment of concept accuracy. The key instructions needed were to guide the evaluator to focus on the presence of relevant entities, and not their values or the output format. The final prompt which includes the instructions, the output of the student model, the gold code and the question are shown in Figure 6 in the Appendix.

We define the concept accuracy as the percentage of cases where the student model output receives a rating of 5. For all models, we measure the concept accuracy for the base model as well as checkpoint after one epoch (see Figure 8–12 in the Appendix for rating distributions). The concept accuracy shown in Table 3, indicates the ORCA-2 family models have significantly lower concept accuracy initially as compared to other models. However, the fine-tuning leads to substantial improvements in these models, leading to a small gap in concept accuracy as compared to other models post fine-tuning. To better understand these results, we manually examined fifty random sample outputs from each of the base models, that were assessed as lacking concept accuracy. We then examined the output of the models for the same samples after one epoch. While the analysis was performed on all models, in the following sections we present the detailed analysis of PHI-3-MINI and ORCA-2-7B models (See Appendix D for many illustrative examples). The analysis of the remaining models reveals similar patterns.

PHI-3-MINI concept accuracy: For the base PHI-3-MINI model, the overall concept accuracy was around 70% with about 16% samples receiving a rating of 1 or 2 by GPT-4. The PHI-3-MINI model responses don't follow a standard structure while answering the question. In a significant number of samples with low concept rating, the base model's response does not provide the formulas/arithmetic steps that are required to answer the question, thus failing to demonstrate concept understanding. About 7% of the samples received a rating of 4, with many of these responses containing minor arithmetic errors, providing formulas with closely related but not correct entities, and other small issues. Most of the base model responses with missing formulas are corrected by training the model for one epoch. Approximately 80% of cases, initially rated 4, are also corrected after one epoch. Several cases where the formula was properly but incorrectly written by the base model remained incorrect even after 1 epoch.

ORCA-2-7B concept accuracy: The initial ORCA-2 model provides a long explanation of the required reasoning, often failing to produce executable code. Sometimes the formulae are written descriptively without use of mathematical representations. In a significant number of cases, the model includes the input passage leading to an incomplete output that does not contain the required formula. As a result, 43% of the samples received a rating of 1 or 2 and the model has a low concept accuracy of 28%. Occasionally, the long explanations do provide the correct formula which is not identified by the GPT-4. These results are likely due to the ORCA-2 being a model that is specialized to solve reasoning problems using elaborate reasoning traces and not being explicitly trained in generating code that encodes the reasoning. Hence the ORCA-2 model tends to produce output that is significantly different from that expected for the

Methods	MSTL-7B⋆	ORCA-7B⋆	ORCA-13B⋆	PHI-3.8B⋆	PHI-14B⋆	GPT-3.5T	GPT-4
FinQA Accuracy							
zero-shot	41.58	2.78	37.49	57.62	70.35	66.52	**77.51**
few-shot†	50.82	15.34	4.36	42.45	66.95	67.39	**78.46**
fine-tuned	76.63	71.57	73.75	73.49	**77.59**	-	-
ConvFinQA Accuracy							
zero-shot	31.82	2.61	28.03	48.21	59.38	67.45	**76.95**
few-shot†	18.52	10.68	11.88	41.33	60.09	65.79	**82.42**
fine-tuned	76.48	70.30	75.77	76.00	**81.94**	-	-
TATQA Accuracy							
zero-shot	66.01	4.59	44.29	78.27	85.37	85.00	**93.00**
few-shot†	66.85	26.88	17.83	59.33	76.32	74.75	**91.89**
fine-tuned	88.71	88.57	88.86	90.94	**93.03**	-	-

⋆ MSTL-7B, ORCA-{x}B, PHI-3.8B, PHI-14B and GPT-3.5T represent MISTRAL 7B, ORCA-2-{x}B, PHI-3-MINI, PHI-3-MEDIUM, and GPT-3.5 TURBO respectively.

† Few-shot PoT prompting is used with 4-shots selected from the few-shots used in (Chen et al., 2023).

Table 2: Accuracy of the fine-tuned models on financial datasets.

Fine-tuned Models	Concept		Entity Extraction		Executable Code	
	Base model	Epoch-1	Base model	Epoch-1	Base model	Epoch-1
MISTRAL 7B	53.09	84.18	66.99	90.72	56.73	99.54
ORCA-2-7B	27.94	76.89	57.54	90.71	10.87	91.95
ORCA-2-13B	46.48	79.6	71.78	92.69	29.67	91.95
PHI-3-MINI	69.22	82.94	82.81	90.23	94.33	96.82
PHI-3-MEDIUM	79.23	84.55	65.26	90.48	98.18	96.94

Table 3: Evaluation of concept, entity extraction and executable code accuracy by GPT-4 for the FinQA dataset

specific problem formulation considered in this study.

Upon fine-tuning, the model quickly learns to produce executable code with the formula written out in the desired format, while suppressing unnecessary output. We observe that the model does correct some of the formula mistakes that it made initially. Similar to PHI-3-MINI, we observed that many cases (12% of the samples) initially had a 4 rating, triggered by minor errors. A vast majority of these cases get corrected during fine-tuning. These improvements are reflected in the fine-tuned ORCA-2 model achieving a significantly improved concept accuracy of 77% after one epoch.

5.2 Entity extraction

To measure the entity extraction capability of the student models, we incorporate the first line of the teacher model's code into the fine-tuning prompt and perform inference on the fine-tuned model to complete the Python code. Since the provided line is a comment with the formula required to answer the question, the main task is to extract the required entities. We then use GPT-4 with the prompt shown in Figure 7 in the Appendix, to assess the student code and determine if all the required entities have been correctly extracted.

The results shown in Table 3 indicate significant improvement in entity extraction capability of all the student models during the fine- tuning, with all of them showing similar accuracy after fine-tuning. The results suggest that the fine-tuning helps the base model adapt to the specific table structures and data format present in the financial dataset, improving the entity extraction performance and contributing to overall model accuracy.

5.3 Code generation

As the required code for this problem is simple, we only use the ability to generate executable code as a measure of accuracy. The results are shown in Table 3. As compared to other models, the base PHI-3 models have very high percentage of cases where they generate executable code. After epoch

Fine-tuned Models	**Training dataset** (samples from train split)			
	FinQA:5698	FinQA:1500	FinQA:1000 ConvFinQA: 500	ConvFinQA:2550
FinQA accuracy				
MISTRAL 7B	76.63	70.35	68.78	64.95
ORCA-2-7B	71.57	63.56	67.13	55.01
ORCA-2-13B	73.75	70.53	70.09	65.13
PHI-3-MINI	73.49	69.83	71.14	68.87
PHI-3-MEDIUM	77.59	74.71	75.06	74.80
ConvFinQA accuracy				
MISTRAL 7B	36.34	33.01	72.20	76.48
ORCA-2-7B	41.09	32.3	69.12	70.30
ORCA-2-13B	36.82	41.81	72.92	75.77
PHI-3-MINI	44.89	49.40	72.20	76.00
PHI-3-MEDIUM	53.44	51.54	79.57	81.94

Table 4: Effect of training data size on fine-tuning: FinQA and ConvFinQA datasets.

Fine-tuned Models	**Training dataset** (samples from train split)			
	FinQA:5698	FinQA:1500	FinQA:1000 TATQA: 500	TATQA:4878
FinQA accuracy				
MISTRAL 7B	76.63	70.35	70.01	56.06
ORCA-2-7B	71.57	63.56	67.48	57.11
ORCA-2-13B	73.75	70.53	68.70	63.55
PHI-3-MINI	73.49	69.83	70.70	63.81
PHI-3-MEDIUM	77.59	74.71	72.44	70.96
TATQA accuracy				
MISTRAL 7B	84.26	78.41	83.98	88.71
ORCA-2-7B	79.38	72.56	80.64	88.57
ORCA-2-13B	80.78	79.94	85.93	88.86
PHI-3-MINI	81.19	74.51	86.07	90.94
PHI-3-MEDIUM	88.30	85.79	88.16	93.03

Table 5: Effect of training data size on fine-tuning: FinQA and TATQA datasets.

1, all models have more than 90% success rate of producing executable code, indicating that the models initially lacking the ability to generate the required code, have significantly improved.

Overall, the detailed assessment revealed two main reasons that leads to improved model performance post fine-tuning: (1) Training with a standard reasoning chain induces the models to express and apply the required concept and (2) Adapting to the specific data format improves the entity extraction performance.

5.4 Effect of training data size

Given the observed effects of fine-tuning, it naturally leads us to inquire whether a smaller dataset might be adequate for achieving similar enhancements. Another related and interesting question is the volume of data necessary to adapt the FinQA model for proficiency with ConvFinQA, which, while originating from the same domain as FinQA, introduces the additional complexity of processing conversational-style questions.

We perform fine-tuning experiments with the following settings to understand the data requirements (a) 1500 data points randomly sampled from the original FinQA dataset and (b) 1000 samples randomly sampled from the FinQA dataset combined with 500 samples randomly sampled from ConvFinQA dataset. The models fine-tuned on these datasets is compared with the corresponding mod-

els trained on the entire FinQA and ConvFinQA training datasets, respectively. Evaluation on the test data of both datasets is shown in Table 4.

The model trained on 1500 data points from FinQA is within 3%-8% of the model trained with the entire data, suggesting that effective fine-tuning can be achieved with a significantly smaller dataset. The models trained on the entire FinQA dataset perform poorly when directly used on the ConvFinQA test data. However, when the models are trained with a smaller dataset consisting of 1000 FinQA samples and 500 samples from ConvFinQA data, there is improvement of more than 25% across models, with all models achieving an accuracy within 5% of the model trained with the entire ConvFinQA training data. These findings suggest that the financial concept understanding along with fine-tuning provides most of the key learning that can address ConvFinQA questions as well. However, the model needs a small sample from the ConvFinQA dataset to adapt to the conversational style of questions used in that dataset.

We perform a similar experiment to examine the performance of models trained on FinQA on TATQA dataset. Unlike the ConvFinQA data which is derived from the FinQA dataset, the TATQA is an independent financial question answering data set. The results of the evaluation are shown in Table 5. The models trained on the complete FinQA train data set perform reasonably on the TATQA test data as compared to the corresponding models trained on the full TATQA training data. The MISTRAL 7B and the PHI-3-MEDIUM are within 5% of the accuracy achieved by the model trained on TATQA data. Some of the models trained on the 1500 FinQA data points show significant and varying decline in performance as compared to those trained on the entire FinQA data set. However, when trained on a combination of 1000 FinQA data points and 500 TATQA data points, all models show excellent performance achieving accuracy within 8% of the model trained on the full TATQA data set. These results demonstrate the potential to efficiently adapt a financial question answering model to a different financial data set.

6 Conclusion

We explored the performance of small language models fine-tuned for financial question answering, using exemplars generated by a very large teacher model. The small models achieved accuracy comparable to that of the teacher model, driven primarily by improved ability to apply financial concepts as well as entity extraction. We showed that smaller datasets can yield similar results, suggesting that small language models can be efficiently fine-tuned for complex financial domain problems.

Limitations

The use of GPT-4 to assess concept understanding using the base and fine-tuned student models' output, can sometimes produce erroneous determinations of concept error. While we instruct GPT-4 to not assess based on the output format, we found that elaborate responses could sometimes lead to a false assessment. Despite this limitation, the method is still effective in achieving our primary goal of understanding the effect of fine-tuning for financial question answering.

While we perform hyperparameter optimization for fine-tuning the student models, the small differences between the performance of the fine-tuned models could be simply due the hyperparameters not being fully optimal. Since we focus more on the improvements achieved in the fine-tuned model over their corresponding base model, this doesn't have a major impact on the findings reported in the paper.

Our experiments are limited to only on the single task of financial question answering. The performance and behaviour of the small models in a multi-task setup needs to be explored in the future.

While we demonstrate that small language models can achieve performance approaching that of much larger models, they also inherit some of the associated risks. For cases where the reasoning was incorrect, the current system will provide an explanation with a high level of confidence, which can be misleading. Our models currently do not address or control for such behavior and we have not studied the nature or extent of this problem. In practice, this can pose challenges for practical use in real world systems. Future research to understand this potential risk in more detail and provide indications of when the model is not sure of its response would be valuable.

Disclaimer

The views reflected in this article are the views of the authors and do not necessarily reflect the views of the global EY organization or its member firms.

References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. 2024. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. *arXiv preprint arXiv:2404.14219*.

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. 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 7319–7328.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, et al. 2021a. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*.

Wenhu Chen. 2022. Large Language Models are few (1)-shot Table Reasoners. *arXiv preprint arXiv:2210.06710*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research*.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting Hao Huang, Bryan Routledge, et al. 2021b. FINQA: A Dataset of Numerical Reasoning over Financial Data. In *2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021*, pages 3697–3711. Association for Computational Linguistics (ACL).

Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. 2022. ConvFinQA: Exploring the Chain of Numerical Reasoning in Conversational Finance Question Answering. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6279–6292.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, et al. 2022. PaLM: Scaling Language Modeling with Pathways. *arxiv:2204.02311*.

Yao Fu, Hao Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. 2023. Specializing Smaller Language Models towards Multi-Step Reasoning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10421–10430. PMLR.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.

Amelia Glaese, Nat McAleese, Maja Trębacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, et al. 2022. Improving alignment of dialogue agents via targeted human judgements. *arXiv preprint arXiv:2209.14375*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. 2023. ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving. *arXiv preprint arXiv:2309.17452*.

Namgyu Ho, Laura Schmid, and Se-Young Yun. 2023. Large Language Models Are Reasoning Teachers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 14852–14882.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes. *arXiv preprint arXiv:2305.02301*.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361*.

Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213. Curran Associates, Inc.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, et al. 2022. Solving Quantitative Reasoning Problems with Language Models. In *Advances in Neural Information Processing Systems*, volume 35, pages 3843–3857. Curran Associates, Inc.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. 2023. Teaching small language models to reason. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1773–1781, Toronto, Canada. Association for Computational Linguistics.

Arindam Mitra, Luciano Del Corro, Shweti Mahajan, Andres Codas, Clarisse Simoes, Sahaj Agarwal, Xuxi Chen, Anastasia Razdaibiedina, Erik Jones, Kriti Aggarwal, et al. 2023. Orca 2: Teaching Small Language Models How to Reason. *arXiv preprint arXiv:2311.11045*.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive Learning from Complex Explanation Traces of GPT-4. *arXiv preprint arXiv:2306.02707*.

Karmvir Singh Phogat, Chetan Harsha, Sridhar Dasaratha, Shashishekar Ramakrishna, and Sai Akhil Puranam. 2023. Zero-Shot Question Answering over Financial Documents using Large Language Models. *arXiv preprint arXiv:2311.14722*.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. 2023. Formal Mathematics Statement Curriculum Learning. In *The Eleventh International Conference on Learning Representations 2023*. OpenReview.net.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know What You Don't Know: Unanswerable Questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789.

Vered Shwartz, Peter West, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Unsupervised Commonsense Question Answering with Self-Talk. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4615–4629.

Pragya Srivastava, Manuj Malik, and Tanuja Ganu. 2024. Assessing LLMs' Mathematical Reasoning in Financial Document Question Answering. *arXiv preprint arXiv:2402.11194*.

Adrian Theuma and Ehsan Shareghi. 2024. Equipping Language Models with Tool Use Capability for Tabular Data Analysis in Finance. *arXiv preprint arXiv:2401.15328*.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. LaMDA: Language Models for Dialog Applications. *arXiv preprint arXiv:2201.08239*.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. 2024. OpenMathInstruct-1: A 1.8 Million Math Instruction Tuning Dataset. *arXiv preprint arXiv:2402.10176*.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2023a. MathCoder: Seamless Code Integration in LLMs for Enhanced Mathematical Reasoning. *arXiv preprint arXiv:2310.03731*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2023b. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations 2023*. OpenReview.net.

Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. 2023c. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. *arXiv preprint arXiv:2305.07922*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022a. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022b. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *arXiv preprint arXiv:2306.05685*.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. 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.

A Fine-tuning: Experimental Details

The student models are fine-tuned using LoRA module of HuggingFace's PEFT library. We experimented with learning rates of $2.5e^{-5}$, $5e^{-5}$, $1e^{-4}$ and LoRA r parameters 64, 256, 512, 1024 for all the models. The optimal hyperparameters used for fine-tuning are outlined in Table 6 and are categorized into three parts:

1. **LoRA parameters**, which comprise of LoRA config parameters of the HuggingFace's PEFT

LoRA parameters	
LoRA_r	512
LoRA_α	1024
LoRA_dropout	0.1
LoRA_bias	none
PEFT task_type	CAUSAL_LM
target_modules	all linear
Training parameters	
epoch	6
batch size	1
gradient accum steps	1
learning rate (MISTRAL)	2.5e^{-5}
learning rate (ORCA-2)	5e^{-5}
learning rate (PHI-3)	1e^{-4}
bf16	True
Inference parameters (vLLM)	
temperature	0
top_p	0.9
max_tokens	1000

Table 6: Hyperparameters used for fine-tuning

library[5].

2. **Training parameters** include config settings of SFTTrainer of the HuggingFace's TRL library[6].

3. **Inference parameters** are associated with the vLLM[7] settings needed for the inference of fine-tuned models.

The additional details on the student models including license and terms of use is provided in Table 7. We use the ORCA-2 models in a manner consistent with their intended use:

As described in the website: https://www.microsoft.com/en-us/research/publication/orca-2-teaching-small-language-models-how-to-reason/, ORCA-2 models were designed for research settings and should be used only for research purposes, and its testing has only been carried out in such environments. It should not be used in downstream applications.

We further fine-tune ORCA-2 and use it only for research purposes, and will not use it for any commercial purposes.

B Fine-tuning and Baseline Prompts

The zero-shot, few-shot and fine-tuning prompts for all datasets are given in Figure 3, Figure 4 and Figure 5 respectively. We use the chat template that comes with the model's tokenizer for formatting prompt and completion into model specific formats. The formatted prompts and completions are then used during supervised fine-tuning of models and their evaluations.

C GPT-4 for Student Model Assessment

We use GPT-4 to quantify the LLMs conceptual understanding and entity extraction capabilities. In performing this analysis, GPT-4 is prompted to identify if the student code's concept or extracted entities are correct or not. The zero-shot prompt for evaluating these capabilities is given in Figure 6 and Figure 7.

D Examples

We illustrate with examples how the student models PHI-3-MINI and ORCA-2-7B improve in various areas through training. These examples display the student models' responses at the base checkpoint, i.e., checkpoint-0, and after training for epoch 1, i.e., checkpoint-6000. The example shown in Figure 13 demonstrates that the code generation capabilities of the ORCA-2-7B model improve significantly after training for epoch 1. The example provided in Figure 14 illustrates that the ORCA-2-7B model's entity extraction capabilities improve with training.

In some instances, the base PHI-3-MINI model fails to generate a structured response and misses concept generation, as shown in Figure 15. However, this is corrected by training the model for an epoch. The example in Figure 16 demonstrates a case where the entity names are not descriptive at the base checkpoint, but this improves with model training. In general, the improvement in complex concepts does not occur with training the model for an epoch, and this observation is true for the PHI-3-MINI model as well, as seen in Figure 17.

E Few-shots for datasets: FinQA, ConvFinQA and TATQA

We use few-shot prompting for code generation by the teacher model (GPT-4). For all datasets, we use 4-shots in our experiments and these 4-shots for FinQA, ConvFinQA and TATQA datasets are listed in Figure 18, Figure 19 and Figure 20 respectively.

[5]https://github.com/huggingface/peft
[6]https://huggingface.co/docs/trl/v0.7.2/en/sft_trainer
[7]https://github.com/vllm-project/vllm

Model Name	Parameters	HuggingFace API	License
MISTRAL 7B	7B	`mistralai/Mistral-7B-Instruct-v0.2`	apache-2.0
ORCA-2-7B	7B	`microsoft/Orca-2-7b`	msr-lic*
ORCA-2-13B	13B	`microsoft/Orca-2-13b`	msr-lic*
PHI-3-MINI	3.8B	`microsoft/Phi-3-mini-128k-instruct`	mit
PHI-3-MEDIUM	14B	`microsoft/Phi-3-medium-128k-instruct`	mit

* msr-lic stands for microsoft-research-license

Table 7: Description of student LLMs used for fine-tuning

Zero-shot prompt: FinQA and TATQA

Read the following passage and then write Python code to answer the question:
###Passage: `text + table`
###Question: ask question?
###Instructions: First, identify entities required to answer the question. Extract the identified entities and store in python variables. Then perform calculations with the entities and strictly store the answer to the python variable "ans". Python code must end after the variable "ans" is defined. Comments must begin with character "#".
###Python

Zero-shot prompt: ConvFinQA

Read the following text and table, and then answer the last question by writing a python code:
###Passage: `text + table`
###Questions: A series of questions of a conversation?
###Last Question: Last question of the conversation?
###Instructions: First, identify entities required to answer the question. Extract the identified entities and store in python variables. Then perform calculations with the entities and strictly store the answer to the python variable "ans". Python code must end after the variable "ans" is defined. Comments must begin with character "#".
###Python

Figure 3: Zero-shot prompts.

Few-shot prompt: FinQA and TATQA

Few-shots
Read the following passage and then write a python code to answer the question:
###Passage: `text + table`
###Question: ask question?
###Python

Few-shot prompt: ConvFinQA

Few-shots
Read the following text and table, and then answer the last question by writing a python code:
###Passage: `text + table`
###Questions: A series of questions of a conversation?
###Last Question: Last question of the conversation?
###Python

Figure 4: Few-shot prompts.

Fine-tuning prompt-completion pairs: FinQA and TATQA

Read the following passage and then write a python code to answer the question:
###Passage: `text + table`
###Question: ask question?
###Instructions: The final answer must be stored in the Python variable "ans" and comments must begin with character "#".
###Python

`GPT-4 code`
###EndPython

Fine-tuning prompt-completion pairs: ConvFinQA

Read the following text and table, and then answer the last question by writing a Python code:
###Passage: `text + table`
###Questions: A series of questions of a conversation?
###Last Question: Last question of the conversation?
###Instructions: The final answer must be stored in the Python variable "ans" and comments must begin with character "#".
###Python

`GPT-4 code`
###EndPython

Figure 5: Fine-tuning prompt-completion pairs.

Concept evaluation prompt

You are an AI assistant. You will be given the definition of an evaluation metric for assessing the financial concept understanding demonstrated by the provided student code. The assessment is based on the provided question and the gold code that provides the true concept. Please note that the student code can be in a very different format and the format difference should be ignored. Please ignore the values of the required entities in your assessment. Entity extraction is a different skill that is not relevant to assess concept understanding. Your job is to compute an accurate evaluation score using the provided evaluation metric. Make sure to explain your answer.
Concept understanding measures how well the student model code demonstrates an understanding of the financial concept illustrated by the gold code. Consider whether the student code is trying to compute the required entity and is it talking about entities relevant to the required computation. Given the student code, the gold code and the question, score the concept understanding demonstrated by the student code between one to five stars using the following rating scale:
`One star:` the student code demonstrates no understanding of the concept to be calculated
`Two stars:` the student code demonstrates limited understanding of the required concept
`Three stars:` the student code demonstrates partial understanding of the required concept
`Four stars:` the student code mostly demonstrates the understanding of the concept illustrated by the gold code but there are minor issues
`Five stars:` the student code demonstrates perfect understanding of the concept illustrated by the gold code.
This rating value should always be an integer between 1 and 5. So the rating produced should be 1 or 2 or 3 or 4 or 5. The result should strictly be written in the following format: {'Explanation': [Think step by step and explain the reason in detail for the rating. Step 1: Analyse the gold code, Step 2: Analyse the student code, Step 3: Evaluate the student code by comparing with the gold code. Step 4: Provide the final rating with a detailed justification.], 'Star rating': [int]}"
Question: `question?`
Gold code: `Teacher's generated code`
Student generated code: `Student's generated code`

Figure 6: Zero-shot prompt for concept assessment using GPT-4.

Entity extraction evaluation prompt

Based on the question and the gold answer, determine if the student has correctly extracted all the relevant entities. Strictly ensure that the entity values are exactly matching.
Question: `question?`
Gold code: `Teacher's generated code`
Student generated code: `Student's generated code`

Figure 7: Zero-shot prompt for entity extraction assessment using GPT-4.

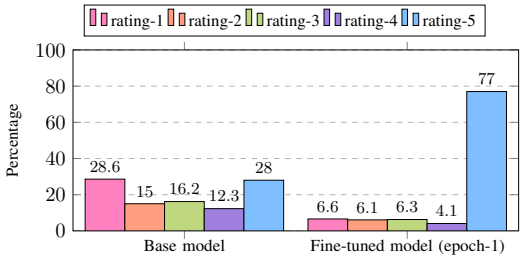

Figure 8: Concept rating for ORCA-2-7B by GPT-4.

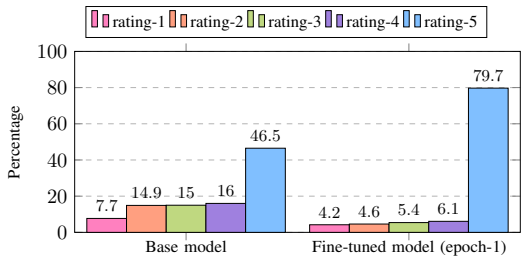

Figure 9: Concept rating for ORCA-2-13B by GPT-4.

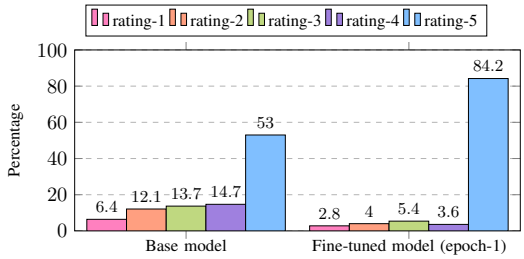

Figure 10: Concept rating for MISTRAL 7B by GPT-4.

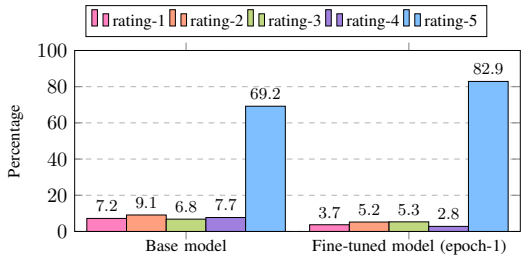

Figure 11: Concept rating for PHI-3-MINI by GPT-4.

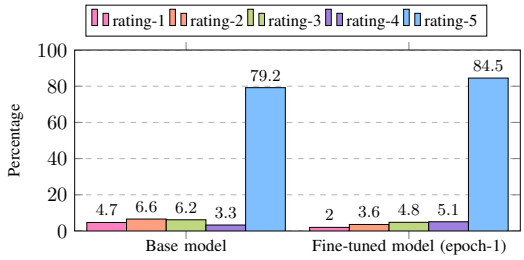

Figure 12: Concept rating for PHI-3-MEDIUM by GPT-4.

IP/2005/page_19.pdf-1

Passage: item 7 . management 2019s discussion and analysis of financial condition and results of operations executive summary international paper 2019s operating results in 2005 were strongly impacted by significantly higher costs for energy . . .
in millions | 2005 | 2004 | 2003
industry segment operating profits | $ 1923 | $ 2040 | $ 1734
corporate items | -597 (597) | -469 (469) | -466 (466)
corporate special items* | -147 (147) | -142 (142) | -281 (281)
interest expense net | -593 (593) | -710 (710) | -705 (705)
accounting changes | 2013 | 2013 | -13 (13)
net earnings (loss) | $ 1100 | $ -35 (35) | $ 302
Question: what was the change in industry segment operating profits between 2004 and 2005?

Gold Program: subtract(1923, 2040)
Gold Answer: -117.0

ORCA-2-7B Checkpoint-0 Code:
To answer the question, we need to find the industry segment operating profits for both 2004 and 2005 and then calculate the change between the two years.
First, let's find the industry segment operating profits for 2004 and 2005:
2004:
Industry segment operating profits = $2040 million
2005:
Industry segment operating profits = $1923 million
Now, let's calculate the change in industry segment operating profits between 2004 and 2005:
Change = Industry segment operating profits in 2005 - Industry segment operating profits in 2004
Change = $1923 million - $2040 million
Change = -$117 million
So, the change in industry segment operating profits between 2004 and 2005 was -$117 million.
Final answer: -$117 million
ORCA-2-7B Checkpoint-0 Executed Answer: `Null`

ORCA-2-7B Checkpoint-6000 Code:
# Calculate: change_in_industry_segment_operating_profits = industry_segment_operating_profits_2005 - industry_segment_operating_profits_2004
industry_segment_operating_profits_2004 = 2040
industry_segment_operating_profits_2005 = 1923
change_in_industry_segment_operating_profits = industry_segment_operating_profits_2005 - industry_segment_operating_profits_2004
ans = change_in_industry_segment_operating_profits
###Final Answer: change_in_industry_segment_operating_profits = 117
ORCA-2-7B Checkpoint-6000 Executed Answer: -117.0

Figure 13: An example illustrating the code generation capabilities of the fine-tuned ORCA-2-7B model.

IP/2005/page_35.pdf-4

Passage: contractual obligations for future payments under existing debt and lease commitments and purchase obli- gations at december 31 , 2005 , were as follows : in millions 2006 2007 2008 2009 2010 thereafter . (a) the 2006 amount includes $ 2.4 billion for contracts made in the ordinary course of business to purchase pulpwood , logs and wood chips . the majority of our other purchase
in millions | 2006 | 2007 | 2008 | 2009 | 2010 | thereafter
total debt | $ 1181 | $ 570 | $ 308 | $ 2330 | $ 1534 | $ 6281
lease obligations | 172 | 144 | 119 | 76 | 63 | 138
purchase obligations (a) | 3264 | 393 | 280 | 240 | 204 | 1238
total | $ 4617 | $ 1107 | $ 707 | $ 2646 | $ 1801 | $ 7657
Question: what was the percent of the total contractual obligations for future payments for total debt in 2005?

Gold Program: divide(1181, 4617)
Gold Answer:0.25579

ORCA-2-7B Checkpoint-0 Code:
To answer the question, we need to find the total contractual obligations for future payments for total debt in 2005 and then calculate the percentage of that amount.
From the passage, we can see that the total debt in 2005 was $1181 million.
Now, let's calculate the percentage:
Percentage = (Total debt in 2005 / Total contractual obligations for future payments for total debt in 2005) * 100
Percentage = ($1181 million / Total contractual obligations for future payments for total debt in 2005) * 100
Percentage = ($1181 / ?) * 100
We don't have the total contractual obligations for future payments for total debt in 2005, so we cannot calculate the percentage.
ORCA-2-7B Checkpoint-0 Executed Answer: `Null`

ORCA-2-7B Checkpoint-6000 Code:
# Calculate: percent_of_total_contractual_obligations_for_future_payments_for_total_debt_2005 = (total_debt_2005 / total_contractual_obligations_2005) * 100
total_debt_2005 = 1181
total_contractual_obligations_2005 = 4617
percent_of_total_contractual_obligations_for_future_payments_for_total_debt_2005 = (total_debt_2005 / total_contractual_obligations_2005) * 100
ans = percent_of_total_contractual_obligations_for_future_payments_for_total_debt_2005
###Final Answer: 23.6%
ORCA-2-7B Checkpoint-6000 Executed Answer: 25.57938

Figure 14: An example illustrating enhanced entity extraction capability with training ORCA-2-7B model.

IP/2007/page_31.pdf-4

Passage: tissue pulp due to strong market demand, particularly from asia. average sales price realizations improved significantly in 2007 . . .
in millions | 2007 | 2006 | 2005
sales | $ 5245 | $ 4925 | $ 4625
operating profit | $ 501 | $ 399 | $ 219
Question: in 2005 what was industrial packaging the profit margin?

Gold Program: divide(219, 4625)
Gold Answer: 0.04735

PHI-3-MINI Checkpoint-0 Code:
# Read the relevant data from the passage
profit_margin_2005 = 219 # The profit margin for industrial packaging in 2005 is given as $219 million

# Store the answer in the variable "ans"
ans = profit_margin_2005
PHI-3-MINI Checkpoint-0 Executed Answer: 219.0

PHI-3-MINI Checkpoint-6000 Code:
# Calculate: profit_margin_2005 = operating_profit_2005 / sales_2005
operating_profit_2005 = 219
sales_2005 = 4625
profit_margin_2005 = operating_profit_2005 / sales_2005
ans = profit_margin_2005
###End Python
PHI-3-MINI Checkpoint-6000 Executed Answer: 0.04735

Figure 15: An example where the concept is not clearly stated (GPT-4 concept rating 1) at checkpoint-0 that improves to GPT-4 concept rating 5 after training PHI-3-MINI for 1 epoch.

SPGI/2018/page_74.pdf-1

Passage: 72 s&p global 2018 annual report. for securities that are quoted in active markets, the trustee/ custodian determines fair value by applying . . .
(in millions) | level 3
balance as of december 31 2017 | $ 39
purchases | 2014
distributions | -2 (2)
gain (loss) | 2
balance as of december 31 2018 | $ 39
Question: what was the ratio of the pension trust assets for 2017 to 2018?

Gold Program: divide(480, 415)
Gold Answer: 1.15663

PHI-3-MINI Checkpoint-0 Code:

```
# First, we extract the pension trust assets for 2017 and 2018 from the passage.

# Pension trust assets for 2017
assets_2017 = 480 # in millions

# Pension trust assets for 2018
assets_2018 = 1739 # in millions

# Now, we calculate the ratio of the pension trust assets for 2017 to 2018.

# Calculate the ratio
ratio = assets_2017 / assets_2018

# Store the result in the variable "ans"
ans = ratio
```

PHI-3-MINI Checkpoint-0 Executed Answer: 0.27602

PHI-3-MINI Checkpoint-6000 Code:

```
# Calculate: ratio_of_pension_trust_assets = pension_trust_assets_2017 / pension_trust_assets_2018
pension_trust_assets_2017 = 1739
pension_trust_assets_2018 = 1572
ratio_of_pension_trust_assets = pension_trust_assets_2017 / pension_trust_assets_2018
ans = ratio_of_pension_trust_assets
###End Python
```

PHI-3-MINI Checkpoint-6000 Executed Answer: 1.10623

Figure 16: An example where the descriptive response (GPT-4 concept rating 5) at checkpoint-0 follows the structured format after training PHI-3-MINI for 1 epoch.

APD/2019/page_31.pdf-2

Passage: part ii item 5 . market for registrant 2019s common equity , related stockholder matters , and issuer purchases of equity securities our common stock . . .
- | 2019 | 2018
first quarter | $ 1.10 | $ .95
second quarter | 1.16 | 1.10
third quarter | 1.16 | 1.10
fourth quarter | 1.16 | 1.10
total | $ 4.58 | $ 4.25
Question: what is the increase observed in the first quarter dividend percentage concerning the total value , during 2018 and 2019?

Gold Program: divide(1.10, 4.58), divide(.95, 4.25), subtract(#0, #1)
Gold Answer: 0.01665

PHI-3-MINI Checkpoint-0 Code:

```
# First, we need to extract the dividend percentages for the first quarter of both 2018 and 2019 from the given data.

# Extracting the first quarter dividend percentage for 2018
first_quarter_2018_dividend_percentage = 1.10 # Given in the passage

# Extracting the first quarter dividend percentage for 2019
first_quarter_2019_dividend_percentage = 1.16 # Given in the passage

# Calculating the increase in the first quarter dividend percentage from 2018 to 2019
increase_in_dividend_percentage = first_quarter_2019_dividend_percentage - first_quarter_2018_dividend_percentage

# Storing the final answer in the variable "ans"
ans = increase_in_dividend_percentage
```

PHI-3-MINI Checkpoint-0 Executed Answer: 0.05999

PHI-3-MINI Checkpoint-6000 Code:

```
# Calculate: increase_in_first_quarter_dividend_percentage = ( first_quarter_dividend_2019 - first_quarter_dividend_2018 ) /
first_quarter_dividend_2018
first_quarter_dividend_2018 = 0.95
first_quarter_dividend_2019 = 1.10
increase_in_first_quarter_dividend_percentage = ( first_quarter_dividend_2019 - first_quarter_dividend_2018 ) / first_quarter_dividend_2018
ans = increase_in_first_quarter_dividend_percentage
###End Python
```

PHI-3-MINI Checkpoint-6000 Executed Answer: 0.15789

Figure 17: An example where the complex concept with GPT-4 concept rating 2 remains unchanged after training PHI-3-MINI for 1 epoch.

Read the following passage and then write python code to answer the question
###Passage: (in millions) | dec 282013 | dec 292012
available-for-sale investments | $ 18086 | $ 14001
cash | 854 | 593
equity method investments | 1038 | 992
loans receivable | 1072 | 979
non-marketable cost method investments | 1270 | 1202
reverse repurchase agreements | 800 | 2850
trading assets | 8441 | 5685
total cash and investments | $ 31561 | $ 26302
###Question: what percentage of total cash and investments as of dec . 29 2012 was comprised of available-for-sale investments?
###Python
#Calculate: percent_available_for_sale_investments_dec_29_2012 = available_for_sale_investments_dec_29_2012 / total_cash_and_investments_dec_29_2012
available_for_sale_investments_dec_29_2012 = 14001
total_cash_and_investments_dec_29_2012 = 26302
percent_available_for_sale_investments_dec_29_2012 = available_for_sale_investments_dec_29_2012 / total_cash_and_investments_dec_29_2012
ans = percent_available_for_sale_investments_dec_29_2012
###End Python</s>

Read the following passage and then write python code to answer the question
###Passage: the chart shows that the firm posted market risk 2013 related gains on 248 out of 261 days in this period , with 12 days exceeding $ 210 million . december 31 (in millions) | 1 basis point increase in jpmorgan chase 2019s credit spread
2010 | $ 35
2009 | $ 39
###Question: on what percent of trading days were there market gains above $ 210 million?
###Python
#Calculate: percent_days_with_market_gains_above_210_million = days_with_market_gains_above_210_million / total_trading_days
days_with_market_gains_above_210_million = 12
total_trading_days = 261
percent_days_with_market_gains_above_210_million = days_with_market_gains_above_210_million / total_trading_days
ans = percent_days_with_market_gains_above_210_million
###End Python</s>

Read the following passage and then write python code to answer the question
###Passage: american tower corporation and subsidiaries notes to consolidated financial statements (3) consists of customer-related intangibles of approximately $ 75.0 million and network location intangibles of approximately $ 72.7 million . the customer-related intangibles and network location intangibles are being amortized on a straight-line basis over periods of up to 20 years.
- | preliminary purchase price allocation
current assets | $ 8763
. . .
fair value of net assets acquired | $ 57536
goodwill (2) | 5998
###Question: for acquired customer-related and network location intangibles , what is the expected annual amortization expenses , in millions?
###Python
#Calculate: amortization_expenses = (customer_related_intangibles + network_location_intangibles) / straight_line_basis
customer_related_intangibles = 75
network_location_intangibles = 72.7
straight_line_basis = 20
amortization_expenses = (customer_related_intangibles + network_location_intangibles) / straight_line_basis
ans = amortization_expenses
###End Python</s>

Read the following passage and then write python code to answer the question
###Passage: the aggregate commitment under the liquidity asset purchase agreements was approximately $ 23.59 billion and $ 28.37 billion at december 31 , 2008 and 2007 , respectively .
(dollars in billions) | 2008 amount | 2008 percent of total conduit assets | 2008 amount | percent of total conduit assets
united states | $ 11.09 | 46% (46 %) | $ 12.14 | 42% (42 %)
australia | 4.30 | 17 | 6.10 | 21
. . .
greece | 0.27 | 1 | 0.31 | 1
other | 1.01 | 5 | 1.26 | 5
total conduit assets | $ 23.89 | 100% (100%) | $ 28.76 | 100% (100%)
###Question: what is percentage change in total conduit asset from 2007 to 2008?
###Python
#Calculate: percent_change_in_total_conduit_assets = (total_conduit_assets_2008 - total_conduit_assets_2007) / total_conduit_assets_2007
total_conduit_assets_2007 = 28.76
total_conduit_assets_2008 = 23.89
net_change_in_total_conduit_assets = total_conduit_assets_2008 - total_conduit_assets_2007
percent_change_in_total_conduit_assets = net_change_in_total_conduit_assets / total_conduit_assets_2007
ans = percent_change_in_total_conduit_assets
###End Python</s>

Figure 18: Few shots for FinQA.

Read the following text and table, and then answer the last question in a series of questions:
###Passage:
- | shares available for awards | shares subject to outstanding awards
2009 global incentive plan | 2322450 | 2530454
2004 stock incentive plan | - | 5923147
###Questions: how many shares are subject to outstanding awards is under the 2009 global incentive plan? what about under the 2004 stock incentive plan? how many total shares are subject to outstanding awards? what about under the 2004 stock incentive plan?
###Last Question: what proportion does this represent?
###Python
# Calculate: shares_outstanding_awards_2009_global_incentive_plan / (shares_outstanding_awards_2009_global_incentive_plan + shares_outstanding_awards_2004_stock_incentive_plan)
shares_outstanding_awards_2009_global_incentive_plan = 2530454
shares_outstanding_awards_2004_stock_incentive_plan = 5923147
total_shares_outstanding_awards = shares_outstanding_awards_2009_global_incentive_plan + shares_outstanding_awards_2004_stock_incentive_plan
proportion = shares_outstanding_awards_2009_global_incentive_plan / total_shares_outstanding_awards
ans = proportion
###End Python</s>

Read the following text and table, and then answer the last question in a series of questions:
###Passage: compensation expense the company recorded $ 43 million , $ 34 million , and $ 44 million of expense related to stock awards for the years ended december 31 , 2015 , 2014 , and 2013 , respectively .
###Questions: what is the compensation expense the company recorded in 2015? what about in 2014? what is the total compensation expense the company recorded in 2015 and 2014? what is the total expenses including 2013?
###Last Question: what is the average for three years?
###Python
# Calculate: average_for_three_years = (compensation_expense_2015 + compensation_expense_2014 + compensation_expense_2013) / 3
compensation_expense_2015 = 43
compensation_expense_2014 = 34
compensation_expense_2013 = 44
total_compensation_expense = compensation_expense_2015 + compensation_expense_2014 + compensation_expense_2013
average_for_three_years = total_compensation_expense / 3
ans = average_for_three_years
###End Python</s>

Read the following text and table, and then answer the last question in a series of questions:
###Passage: the net loss on disposal of those assets was $ 344000 for 2005 and $ 43000 for 2004 .
###Questions: what was the net loss on disposal of assets in 2005? what was the value in 2004? what was the change in value?
###Last Question: what was the percent change?
###Python
# Calculate: percent_change = (net_loss_on_disposal_of_assets_2005 - net_loss_on_disposal_of_assets_2004) / net_loss_on_disposal_of_assets_2004
net_loss_on_disposal_of_assets_2005 = 344000
net_loss_on_disposal_of_assets_2004 = 43000
net_change_in_value = net_loss_on_disposal_of_assets_2005 - net_loss_on_disposal_of_assets_2004
percent_change = net_change_in_value / net_loss_on_disposal_of_assets_2004
ans = percent_change
###End Python</s>

Read the following text and table, and then answer the last question in a series of questions:
###Passage: location | operations conducted | approximatesquare feet | leaseexpirationdates
dublin ireland | global supply chain distribution and administration offices | 160000 | owned
athlone ireland | commercial research and development manufacturing | 80000 | owned
bogart georgia | commercial research and development manufacturing | 70000 | owned
smithfield rhode island | commercial research and development manufacturing | 67000 | owned
###Questions: what is the square feet of the owned global supply chain distribution and administration offices? what is the square feet of the owned commercial research and development manufacturing? what is the sum of those values? what is the total sum including square feet of commercial research and development manufacturing in bogart, georgia? what is the total sum including square feet of commercial research and development manufacturing in smithfield, rhode island?
###Last Question: what is the total sum of square feet owned?
###Python
# Calculate: owned_global_supply_chain_distribution_dublin + commercial_research_and_development_manufacturing_athlone + commercial_research_and_development_manufacturing_bogart + commercial_research_and_development_manufacturing_smithfield
owned_global_supply_chain_distribution_dublin = 160000
commercial_research_and_development_manufacturing_athlone = 80000
commercial_research_and_development_manufacturing_bogart = 70000
commercial_research_and_development_manufacturing_smithfield = 67000
total_square_feet_owned = owned_global_supply_chain_distribution_dublin + commercial_research_and_development_manufacturing_athlone + commercial_research_and_development_manufacturing_bogart + commercial_research_and_development_manufacturing_smithfield
ans = total_square_feet_owned
###End Python</s>

Figure 19: Few shots for ConvFinQA.

Read the following passage and then write python code to answer the question
###Passage: 17. Income Taxes
Income before income taxes for the Company's domestic and foreign operations was as follows:
— | — | Years Ended June 30, | —
($ in millions) | 2019 | 2018 | 2017
Domestic | $204.2 | $140.3 | $56.0
Foreign | 11.8 | 19.9 | 14.2
Income before income taxes | $216.0 | $160.2 | $70.2
###Question: What was the change in Foreign in 2019 from 2018?
###Python
# Calculate: change_in_foreign = foreign_in_2019 - foreign_in_2018
foreign_in_2018 = 19.9
foreign_in_2019 = 11.8
ans = foreign_in_2019 - foreign_in_2018
###End Python</s>

Read the following passage and then write python code to answer the question
###Passage: 11 Intangible assets (continued) (a) Intangible assets RIGHTS AND LICENCES Certain licences that NEXTDC possesses have an indefinite useful life and are carried at cost less impairment losses and are subject to impairment review at least annually and whenever there is an indication that it may be impaired.
. . .
— | Rights and licenses | Internally generated software | Software under development | Total
At 30 June 2019 | — | — | — | —
Cost | 13 | 12,961 | 16,284 | 29,259
Accumulated amortisation | - | -5,580 | - | -5,580
At 30 June 2018 | — | — | — | —
Cost | 104 | 9,555 | 6,509 | 16,168
Accumulated amortisation | -91 | -3,170 | - | -3,261
Net book amount | 13 | 6,385 | 6,509 | 12,907
###Question: Which year have greater total accumulated amortisation?
###Python
# Calculate: find total accumulated amortization and choose year with maximum accumulated amortisation
total_accumulated_amortisation = {'2019': 5580, '2018': 3261}
ans = sorted(total_accumulated_amortisation.items(), key=lambda tup: tup[1], reverse=True)[0][0]
###End Python</s>

Read the following passage and then write python code to answer the question
###Passage: The following table sets forth the breakdown of revenues by category and segment.
. . .
Year Ended December 31, | — | —
— | 2019 | 2018
Total Asia Pacific revenues | 6,490 | 7,859
Total Europe revenues | 36,898 | 36,149
Total North America revenues | 68,024 | 67,314
Total revenues | $111,412 | 111,322
###Question: In 2019, how many geographic regions have total revenues of more than $20,000 thousand?
###Python
# Calculate: find locations with revenue more than $20,000 in a list and count
total_revenues_in_all_regions = {'Asia Pacific': 6490, 'Europe': 36898, 'North America': 68024}
regions_have_more_than_20000_thousand_total_revenues = [k for k, v in total_revenues_in_all_regions.items() if v > 20000]
ans = len(regions_have_more_than_20000_thousand_total_revenues)
###End Python</s>

Read the following passage and then write python code to answer the question
###Passage: Effective Income Tax Rate A reconciliation of the United States federal statutory income tax rate to our effective income tax rate is as follows: In 2019 and 2018 we had pre-tax losses of $19,573 and $25,403, respectively, which are available for carry forward to offset future taxable income.
— | Year Ended | Year Ended
— | December 31, 2018 | December 31, 2019
United States federal statutory rate | 21.00% | 21.00%
State taxes, net of federal benefit | 1.99% | -0.01%
###Question: What was the 2019 percentage change in pre-tax losses?
###Python
# Calculate: percentage_change_in_pre_tax_losses = (pre_tax_losses_2019 - pre_tax_losses_2018) / pre_tax_losses_2018 * 100
pre_tax_losses_2018 = 25403
pre_tax_losses_2019 = 19573
net_change = pre_tax_losses_2019 - pre_tax_losses_2018
ans = net_change / pre_tax_losses_2018 * 100
###End Python</s>

Figure 20: Few shots for TATQA.