# Building a Financial Question-Answering App with Fine-Tuned LLMs

A screenshot of a computer

AI-generated content may be incorrect.

## **Project Description**

This project aims to build a basic AI-powered application that answers financial questions based on uploaded documents. The goal is to interpret tables, text, and numerical data in complex financial reports using a large language model (LLM). The pipeline involves dataset preparation, model fine-tuning, inference integration, and evaluation.

## Data Preparation

The train.json dataset includes question-answer pairs alongside with pre-text, post-text, and tables. The original structure was not suitable for direct model training. I reformatted the dataset into a clean structure:

{

**"prompt":** "Question: what was the percent of the growth in the revenues from 2007 to 2008\n\nContext:\nTable:\n...\nPre-text:\n...\nPost-text:\n...\nSteps:\n1. subtract(9362.2, 9244.9) = 117.3\n2. divide(#0, 9244.9) = 1.3%\nAnswer: 1.3%",

**"question":** "what was the percent of the growth in the revenues from 2007 to 2008",

**"answer":** "1.3%"

}

This structured prompt helped the model understand both context and reasoning steps.

# Models

### **Model 1: EleutherAI/gpt-neo-1.3B**

The first model explored was EleutherAI/gpt-neo-1.3B.

**Reasons for Choosing:**

* **Open-source and Accessible:** This model is freely available and relatively easy to implement.
* **Moderate Size:** With 1.3 billion parameters, it represents a balance between model capacity and computational resources required for training and inference. It was hypothesized to be large enough to understand some financial context but still manageable on available hardware.
* **Established Architecture:** GPT-neo is based on the Transformer architecture, which has proven effective for various natural language processing tasks.

**Expected Limitations and Fallbacks:**

* **Limited Financial Domain Knowledge:** A general-purpose model like gpt-neo-1.3B is not pre-trained specifically on financial documents and terminology. This could lead to difficulties in understanding nuanced financial questions.
* **Potential for Hallucinations:** Without specific fine-tuning on financial data, the model might generate incorrect or irrelevant answers.
* **Difficulty with Complex Reasoning:** Financial questions often require multi-step reasoning and understanding of relationships between different data points, which might be challenging for a model of this size without targeted training.

**Observations:**

I initially tried zero-shot prompting with gpt-neo-1.3B.

**Example of Zero-Shot Prompting:**

Prompt: You are a financial assistant. Answer the question with numbers only.

This returns answers such as:  
“The answer is: The answer is: The answer” or “The percentage change in equipment rents is ba...”

To improve the model's performance, a few-shot prompting technique was then explored. This involved providing the model with one or more examples of the desired question-answer format, along with the relevant context.

**Example of Few-Shot Prompting:**

Prompt: You are a financial assistant. Answer the question with numbers only.

Question: What was the net income in 2022?

Context: ... (Financial table for 2022) ...

Answer: $109000

After that returns:  
“The percentage change in cash provided by operating activities from”  
  
 noted, EleutherAI/gpt-neo-1.3B has been observed to be "stubborn" and not always follow prompt instructions precisely. This aligns with observations in the community regarding its instruction-following capabilities compared to more recent models.  
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Reference:** While a specific academic paper might not explicitly state "stubborn," various discussions and evaluations within the NLP community highlight that earlier GPT-style models can be less adept at adhering to complex or specific prompt formats compared to models fine-tuned for instruction following.

**2.2 Mistral-7B**

The second model explored was Mistral-7B.

**Reasons for Choosing:**

* **Strong Performance for its Size:** Mistral-7B has demonstrated impressive performance on various benchmarks, often outperforming larger models, indicating a more efficient architecture and training.
* **Good Instruction Following Capabilities:** Mistral models are known for their ability to understand and follow instructions effectively, which is crucial for a question-answering task.
* **Open-Source and Relatively Accessible:** While larger than gpt-neo-1.3B, it is still manageable with appropriate hardware and widely available.

**Expected Limitations and Fallbacks:**

* **Still Requires Financial Domain Adaptation:** Although a strong general-purpose model, it lacks specific pre-training on financial data, potentially limiting its understanding of complex financial concepts without fine-tuning.
* **Potential for Context Window Limitations:** Like other Transformer models, Mistral-7B has a finite context window, which might restrict its ability to process very long financial documents effectively.
* **Computational Cost:** Fine-tuning a 7 billion parameter model requires more computational resources and time compared to a 1.3 billion parameter model.

**Observations:**

Fine-tuning Mistral-7B using the cleaned train.json file resulted in correct answers in the user interface (UI) when tested with uploaded documents. This suggests the model learned to extract and process information from the provided context. However, the evaluation on the dev.json file still yielded very low accuracy. This discrepancy indicates a potential issue with the evaluation methodology or the model's generalization to slightly different data distributions within the development set.

**2.3 tiny-Llama**

The final model explored was tiny-Llama.

**Reasons for Choosing:**

* **Small Footprint and Efficiency:** As a significantly smaller model, tiny-Llama offers the potential for faster inference and reduced computational requirements, making it attractive for deployment on resource-constrained environments.
* **Llama Architecture:** Based on the Llama architecture, it benefits from advancements in model design and training techniques.

**Expected Limitations and Fallbacks:**

* **Limited Capacity:** Due to its small size, tiny-Llama might have a limited capacity to understand complex financial relationships and perform multi-step reasoning.
* **Greater Need for Domain-Specific Fine-tuning:** Its smaller size likely necessitates more targeted and extensive fine-tuning on financial data to achieve satisfactory performance.
* **Potential for Information Loss:** The aggressive downscaling might lead to a loss of some general knowledge and reasoning abilities present in larger models.

**Observations:**

Similar to Mistral-7B, tiny-Llama showed correct answers in the UI with external documents after fine-tuning. However, the evaluation on the cleaned dev.json file again resulted in low accuracy. This reinforces the hypothesis that the issue might lie in the evaluation process or a subtle difference between the training and development datasets.

**3. Model Training and Fine-tuning**

The models were trained and fine-tuned using **Low-Rank Adaptation (LoRA)**. LoRA is a parameter-efficient fine-tuning technique that freezes the pre-trained model weights and introduces a small number of trainable rank-decomposition matrices to each layer of the Transformer architecture. This significantly reduces the number of trainable parameters, leading to faster training times and lower memory consumption, which is particularly beneficial when using GPUs with limited memory, such as the A100 in a Colab notebook environment. The training process was conducted on Google Colaboratory utilizing A100 GPUs, which provided the necessary computational power for efficient fine-tuning.

**4. Evaluation Metrics and Results**

The performance of the fine-tuned models was evaluated using the following metrics:

* **F1-Score:** The F1-score is the harmonic mean of precision and recall.
  + **Precision:** The proportion of correctly predicted positive outcomes out of all predicted positive outcomes.
  + **Recall:** The proportion of correctly predicted positive outcomes out of all actual positive outcomes.
  + For this task, a prediction was considered "positive" if the model's answer matched the ground truth. The F1-score provides a balanced measure, especially when dealing with imbalanced datasets (though this might not be the primary concern here). It is calculated as:

Code snippet

F\_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}

* **Exact Match (EM):** This is a strict metric that measures the percentage of predictions that exactly match the ground truth answer, including capitalization and punctuation. A prediction is either completely correct (1) or completely incorrect (0). The EM score is the average of these binary scores over the evaluation set.
* **Numeric Close:** This metric is specifically designed for evaluating numerical answers. A predicted numerical answer is considered correct if it is within a certain tolerance (absolute or relative) of the ground truth numerical answer. For example, if the ground truth is 100 and the tolerance is 5%, predictions between 95 and 105 would be considered correct. The calculation involves comparing the absolute difference or the percentage difference between the prediction and the ground truth to a predefined threshold.
* **Mean Absolute Percentage Error (MAPE):** MAPE measures the average absolute percentage difference between the predicted and actual 1 numerical values. It is calculated as:

[1. github.com](https://github.com/DATAPAVAN/machine_learning" \t "_blank)

[github.com](https://github.com/DATAPAVAN/machine_learning" \t "_blank)

Code snippet

\text{MAPE} = \frac{1}{n} \sum\_{i=1}^{n} \left| \frac{\text{Actual}\_i - \text{Predicted}\_i}{\text{Actual}\_i} \right| \times 100\%

where n is the number of data points. MAPE provides an easily interpretable measure of the relative error.

* **Symmetric Mean Absolute Percentage Error (SMAPE):** SMAPE is a variation of MAPE that addresses the asymmetry issue where the percentage error can be different depending on whether the prediction overshoots or undershoots the actual value. It is calculated as:

Code snippet

\text{SMAPE} = \frac{1}{n} \sum\_{i=1}^{n} \frac{|\text{Actual}\_i - \text{Predicted}\_i|}{(|\text{Actual}\_i| + |\text{Predicted}\_i|) / 2} \times 100\%

SMAPE provides a more symmetric measure of the percentage error and has upper and lower bounds (0% to 200%).

The specific implementation of these metrics likely involved iterating through the dev.json file, feeding the questions and context to the fine-tuned models, obtaining the predicted answers, and then comparing these predictions to the ground truth answers based on the criteria defined for each metric. For numerical answers, a process to extract and compare the numerical values from both the predicted and ground truth strings would have been necessary.

A graph comparing the performance of the three models (gpt-neo-1.3B, Mistral-7B, tiny-Llama) across these evaluation metrics would visually summarize the results.

**5. Reasons for Low Accuracy and Exact Matches**

The consistently low accuracy and exact match scores on the dev.json file, despite seemingly correct answers in UI testing with uploaded documents, could be attributed to several factors:

* **Subtle Differences Between Training and Development Data:** While the data preparation steps aimed to create a consistent format, there might be subtle variations in the structure, complexity, or the way questions are phrased in the dev.json file compared to the data used for UI testing (which might be a subset of the training data or slightly different in format).
* **Overfitting to the Training Data:** The models might have overfit the specific patterns and phrasing in the train.json file, leading to poor generalization on the dev.json data, even if the underlying financial concepts are similar.
* **Strictness of Exact Match:** The exact match metric is very strict. Even minor differences in formatting, punctuation, or the inclusion of extra words can lead to a prediction being marked as incorrect, even if the core numerical answer or the factual information is correct.
* **Complexity of Financial Questions:** Financial questions can be nuanced and require precise answers. The models might be struggling with the level of precision required, leading to answers that are close but not exact.
* **Evaluation Script Issues:** There might be an issue in the evaluation script itself, such as incorrect parsing of the predicted or ground truth answers, or overly strict comparison criteria (especially for numerical answers if "numeric close" is not being applied correctly).
* **Lack of Robust Numerical Extraction and Comparison:** If the evaluation relies heavily on exact string matching, numerical answers that are semantically correct but formatted slightly differently (e.g., "$1,000" vs. "1000.00") would be marked as incorrect. Implementing a robust numerical extraction and comparison mechanism (as used in the "numeric close" metric) is crucial for financial QA.
* **Context Understanding Limitations:** While the models might be able to extract information when the relevant data is clearly presented, they might struggle with more complex questions that require integrating information from different parts of the context or performing multi-step reasoning.

Further investigation into the dev.json file and the evaluation script is necessary to pinpoint the exact reasons for the low accuracy and exact match scores. Analyzing specific examples where the UI testing yielded correct results but the evaluation failed could provide valuable insights.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.