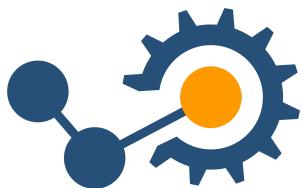


INSTITUTE FOR VISUALISATION AND INTERACTIVE
SYSTEMS

HUMAN-COMPUTER INTERACTION DEPARTMENT



UNIVERSITY OF STUTTGART

70569, STUTTGART

RESEARCH PROJECT REPORT

**CryptoLLM - Large Language Models for
Cryptography Assistance**

MANPA BARMAN

Course of Study: MSc in Information Technology (INFOTECH)

Examiner: Prof. Dr. Andreas Bulling

Supervisor: Mayar Alfares

Commenced : 17/11/2023

Completed: 17/05/2024

Declaration

I hereby declare that the work presented in this research project report is entirely my own. I did not use any other sources and references other than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

Signature

Date

Abstract

Real-life mathematical problems require a combination of natural language understanding, logical thinking, and computational skills. Cryptography is one such complicated field of computer science that requires a good blend of natural understanding and mathematical reasoning skills. Current tools for solving such problems are limited in their ability to either understand and interpret only natural language or only excel in mathematical computations if inputted in a specific format. Large Language Models (LLMs) have shown great promise in solving a wide range of problems by understanding and generating human-like text. However, the currently popular LLMs like GPT-3.5 [1], llama series [2] etc., show suboptimal performance in solving mathematical problems. It is mostly due to the lack of specialized training data and the design of arithmetic problems which has a single correct answer with multistep reasoning. In this project, we design a LLM-based assistant for cryptography problems. We also propose a novel crypto dataset, **CryptoQA** which is curated from popular academic textbooks and resources and is publicly available. Four versions of finetuned models (**CryptoLLMs**) are also trained for solving cryptographic problems. The first two models are fine-tuned on an already math-tuned and a general-purpose LLM respectively using the CryptoQA dataset. The other two versions are fine-tuned in two stages - first on a publicly available math dataset and then on the CryptoQA dataset. A qualitative behavioral analysis of the fine-tuned models is conducted and is made publicly available for experimentation and research.

Contents

1	Introduction	11
1.1	Background	11
1.2	Problem Statement	13
1.3	Research Objectives and Contributions	13
1.4	Scope of the Work	14
2	Literature Review	15
2.1	Existing Mathematical Reasoning-based Datasets	15
2.2	Large Language Models for Cryptography	15
2.3	Data Augmentation Techniques	16
2.4	Fine Tuning	16
3	Methodology	19
3.1	Dataset Preparation	19
3.1.1	Task and Scope of the Dataset	19
3.1.2	Data Collection and Source	20
3.1.3	Cleaning and Preprocessing	21
3.1.4	Data Annotation	24
3.1.5	Data Augmentation	25
3.1.6	Data Format Conversion	27
3.1.7	CryptoQA Dataset	29
3.2	Fine Tuning	29
3.2.1	Techniques used	29
3.2.2	Choice of the Large language model for fine tuning	34
3.2.3	Approach	34
4	Results and Discussion	39
4.1	Experimental Setup	39
4.1.1	Hardware Setup	39
4.1.2	Software Setup	39
4.2	LLM Training Performance	39
4.2.1	Loss Curves	39
4.2.2	Two stage training performance	41
4.2.3	Qualitative Model Performance - Ablation Study	42
4.3	Evaluation and Analysis	47
4.3.1	How the current models perform?	47
4.3.2	Evaluation techniques in large language models	48
4.3.3	Significance of the Loss Curves	48
4.3.4	Analysis with chaining in large language models	49
4.3.5	Analysis using prompt engineering techniques	50
4.3.6	Can LLM learn a textbook?	52

4.3.7	Can RAG solve this issue of crypto chatbot?	53
5	Conclusion	55
5.1	Summary of Findings	55
5.2	Limitations	55
5.3	Future Work	57
Appendices		63
A	Sample Data Examples	63
B	Prompts	63
C	Hugging Face Dataset and Models	64

List of Figures

1.1	The rise of AI-based LLMs; Made with VizSweet	12
1.2	SOTA LLMs response on conceptual cryptography concepts	14
3.1	Illustration of Tokenization in LLM [3]	21
3.2	Chain of Thought approach used in CryptoQA	23
3.3	Forward reasoning sample from CryptoQA	25
3.4	Backward reasoning sample (Self Verification Question) from CryptoQA	26
3.5	Backward reasoning sample (FOBAR question) from CryptoQA	26
3.6	Value Modification in CryptoQA	27
3.7	Rephrasing in CryptoQA	28
3.8	Sample of a dataset in CryptoQA	29
3.9	Percentage Distribution of types in CryptopQA	30
3.10	Distribution of categories in the CryptoQA dataset	30
3.11	Percentage Distribution of topics in CryptoQA	31
3.12	LoRA reparameterization; Only A and B matrices are trained [4]	32
3.13	Fine Tuning Pipeline	35
4.1	Loss Curve Characteristics of Llama and Mistral Mathematics tuned models	40
4.2	Learning Curve Characteristics of Llama and Mistral Mathematics tuned models	40
4.3	Loss Curve Characteristics of Math tuned models on the CryptoQA dataset	40
4.4	Response of Llama 2 7b chat model on basic Caesar Cipher	48
4.5	Illustration of how language chaining works in an LLM	49
4.6	Language chaining in numerical questions	49
4.7	Language chaining in mixed logic questions	51
5.1	Cost of Training GPT models in MosaicML Cloud	56
2	Sample of a cryptographic word problem	63
3	Sample of a number theory math problem in CryptoQA	63
4	Sample of a cryptographic cipher problem in CryptoQA	63
5	System Prompt used for dataset generation	64
6	Rephrasing Prompt used for dataset augmentation	65
7	Value Modification Prompt used for dataset augmentation	66
8	Backward Reasoning Prompt used for dataset augmentation	67
9	Hugging Face Dataset	68

Chapter 1

Introduction

1.1 Background

Cryptography is a very complicated field of computer science. Due to the inherent complexity, which forms the basis of security of our digital communications and transactions, its practical application often faces limitations. It heavily relies on strong mathematical foundations, in well-established areas of mathematics like number theory, probability theory, and abstract algebra, which are fundamentally similar to traditional mathematical concepts. However, cryptography applies these concepts in very specific ways to achieve its goals of secure communication and data protection. For example, it might use number theory to create algorithms for public-key encryption or probability theory to analyze the security of a particular cryptosystem based on its randomness. Existing tools are primarily limited to basic mathematical computational tools.

While cryptography safeguards communication through complex mathematical algorithms as stated, LLMs have the potential to revolutionize how we interact with these algorithms. LLMs [2, 1, 5, 6, 7, 8], have made a remarkable impact in the past few years in solving downstream natural language processing (NLP) tasks such as text comprehension and generation. These models are based on the transformer architecture [9] which excels at processing and understanding sequential data like text. The transformer architecture is an encoder-decoder-based architecture with self-attention layers which outperforms the traditional recurrent neural network's (RNNs) ability to process information in parallel, capture long-range dependencies, and offer some level of interpretability making it a superior choice for many NLP tasks.

Currently, the LLMs are trained on a huge corpus of unsupervised data, mainly trained on text scraped from corpora of automatically crawled web pages like the common crawl, augmented by more carefully curated data. The GPT3 models, for example, are trained mostly on the web (429 billion tokens), textbooks (67 billion tokens) and Wikipedia pages (3 billion tokens). These require several days of training to obtain the *pre-trained* model. Training these models necessitates vast computational resources, specifically thousands of graphical processing units (GPUs), owing to its huge data, complicated architecture, and massive number of parameters. The financial burden is substantial, with GPU costs alone reaching millions of dollars. OpenAI's technical overview for GPT-3, for instance, revealed training runs requiring GPUs valued at a minimum of \$5 million [10] for the commercial models out in the market.

Along with the closed LLMs like GPT-4 [5], PaLM [11], the open source LLMs like the llama 2 [2], the Mistral series [6], and the newly released Llama 3 [8] are also showing comparable performance in this domain. This stage is referred to as *pre-training stage* and the models released are called the *pre-trained* models. For consumer and academic settings, these models are only fine-tuned based on our downstream tasks due to the

computational constraints.

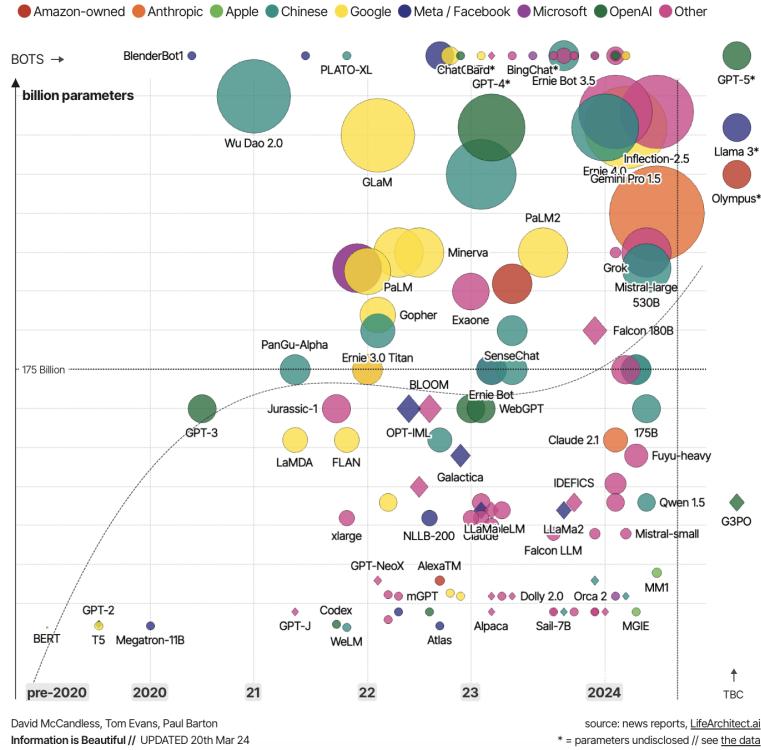


Figure 1.1: The rise of AI-based LLMs; Made with [VizSweet](#)

The LLMs other than answering questions, summarizing documents, translating languages, and completing sentences have demonstrated improved performance on multi-dimensional abilities like coding assistance [12, 13], mathematical reasoning [14, 15, 16, 17] and even video generation [18]. However, the mathematical reasoning tasks have shown sub-optimal performance compared to the other generative solutions and have been considered one of the most challenging domains for these models due to their complicated multi-step reasoning structure of solving. Solving these problems involves understanding natural language input, recalling relevant prerequisite knowledge, performing mathematical computations, and generating precise mathematical expressions. Thus, 'hallucinations' that typically occurs in one step, can propagate to the later steps, eventually leading to a wrong answer [19]. Therefore, quantitative reasoning problems serve as a valuable area for testing the capabilities of language models and as a testbed for developing robust quantitative reasoning solvers that can assist humans in scientific and technical fields and enhance model versatility.

Although the state-of-the-art closed models [5] have recently shown improved performance in mathematical reasoning, it is still a challenge to fine-tune the open-source pre-trained models with better mathematical problem-solving skills. This can be achieved by either of the two popularly adopted methods: *fine tuning* and *prompt-based* methods [20, 21], where both methods share a common goal of improving the accuracy and relevance of reasoning answers generated by LLMs. Fine-tuning involves directly changing the model parameters with a task-specific dataset. Generally, this dataset is much smaller than what is used in the pre-training stage. On the other hand prompt-based methods usually explore various prompting styles and techniques to generate the most desirable response for the given purpose and context. These methods do not tamper with the model parameters but rather try to narrow down the solution space of the LLMs by prompting them with very specific questions. Another commonly used technique that utilizes

prompt engineering techniques on retrieved information from external data sources, such as Wikipedia or domain-specific corpora, is called retrieval augmented generation (RAG). This technique is mostly used in industries where up-to-date information is important, such as in news-related queries or customer chatbot applications.

Fine-tuning is, however, mostly used and preferred for improving the mathematical reasoning capability of the language model as this task demands achieving peak accuracy and fosters a deeper understanding of mathematical concepts. It is also believed that the training dataset used for the pre-trained LLMs has also embedded very basic mathematical notation and problem-solving approaches, which will help the fine-tuning process further. RAG and prompt engineering techniques, on the other hand, might struggle to capture the full complexity of mathematical relationships.

1.2 Problem Statement

Our work aims to focus on the *specialized mathematical domain* of cryptography and create an LLM assistant chatbot by fine-tuning an open-source LLM. This assistant model will solve *mathematical* as well as *conceptual* problems involving cryptographic primitives, which remain unexplored in the literature.

This research proposes a novel approach to address this challenge. By employing open-source LLMs and fine-tuning them with a specialized cryptography dataset, a more accessible and efficient environment for cryptographic tasks will be created. This fine-tuning process will allow for customization to the specific requirements of the task, which can yield highly competitive results without the substantial expenses associated with training and deploying larger models [5]. This innovation has the potential to significantly benefit both users and researchers. On the one hand, users will gain access to AI-powered assistants that can comprehend and explain complex cryptographic concepts, simplifying their interaction with this crucial field. Researchers, meanwhile, can leverage these LLMs to automate repetitive tasks, accelerating the pace of discovery and innovation within cryptography. Research by [22] and [23] have shown that training on domain-specific datasets on mathematical problems can significantly improve the performance of LLMs in the domain of quantitative reasoning.

Recent LLMs perform well for low-difficulty ciphers but suffer in medium and higher-difficulty challenges [24, 25, 26]. Also, it struggles with giving a conceptual understanding of basic cryptographic concepts, as shown in Figure 1.2. Additionally, these studies are mostly conducted with the closed and commercial level LLM whose underlying models and datasets used are unknown to us.

To date, no systematic investigation has been considered to study the impact of fine-tuning different LLMs for the domain of cryptography, which includes a plethora of encryption and decryption techniques. Currently, this space is only explored by prompt-based techniques on the freely available and popular LLMs like OpenAI ChatGPT, Gemini, and Microsoft Bing, and only evaluated on low-difficulty Capture-the-flag challenge questions and university exam questions. Fine-tuning open-source models are not yet explored in this domain of study.

Additionally, there are no benchmark datasets currently for evaluating and even training a cryptographic model, which researchers are actively exploring to establish more robust evaluation methods for cryptographic models.

1.3 Research Objectives and Contributions

In this research we contribute the following:

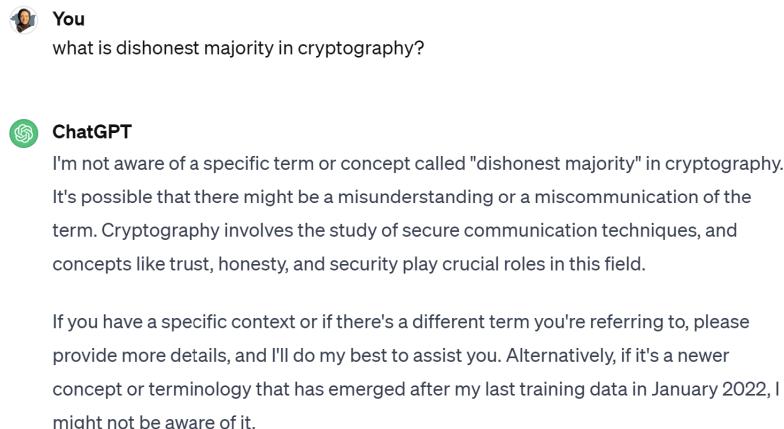


Figure 1.2: SOTA LLMs response on conceptual cryptography concepts

- We propose **CryptoQA** - a novel cryptographic dataset which currently has 510 samples generated with 350 *original questions* and 196 *augmented samples*.
- Based on the **CryptoQA** dataset, we finetuned four 7B parameter models – **CryptоЛLМ**, based on various parameters, with `llama2-7b-chat` and `mistral-7b` as the base pre-trained versions. An extensive analysis was conducted on the behavior of the models under different circumstances.
- An analysis is conducted on the performance of the CryptоЛLМ trained on the **CryptoQA dataset**, with various prompt engineering techniques, language chaining and vulnerabilities to prompt hacking scenarios.

1.4 Scope of the Work

The focus of this study will solely lie in the systematic evaluation of the performance of various language models on cryptographic concepts.

Due to limitations in computational resources and cryptography domain expertise, the current scope of this research is restricted to investigating and evaluating open-source LLMs with parameter sizes of models below 13 billion. As of April 2024, the dataset employed for this research is 510 samples. It is acknowledged that expanding the dataset size has the potential to enhance the quality of the results further.

The focus of the current CryptoLLM-QA dataset is limited to:

- Conceptual definitions of cryptographic concepts.
- Cryptographic problems, categorized as low to medium difficulty based on the training data.

The project envisions future expansion by further augmenting the dataset and incorporating other sources for curating the dataset. This extension aims to ultimately develop the CryptоЛLМ assistant chatbot, which will be equipped to address complex cryptographic reasoning problems.

Chapter 2

Literature Review

2.1 Existing Mathematical Reasoning-based Datasets

LLMs [2, 1, 5, 11, 19, 6] have shown considerable performance in natural language comprehension tasks like summarization, text completion, etc. However, these models have not shown equivalent performance for the mathematical tasks. These models have undergone testing only on basic-level mathematical reasoning datasets such as MATHQA [27](28.5k samples) and GSM8K [28](8.8k samples) or competition-level datasets like MATH [29] (7.5k samples). This limitation may be attributed to the absence of more domain-specific natural language mathematics datasets. The dataset MMLU (Measuring Massive Multi-task Language Understanding) [15] is another popular benchmark that is used in several LLM evaluation tasks. MMLU has a separate math category comprising multiple-choice questions across four mathematics categories. There exist even math datasets in Chinese that consist of questions from the GAOKAO examination, namely GAOKAO-Bench [30]. Similar other Chinese datasets include AGIEval [31], TAL-SCQ5K [32] and MATH23k [33].

All these datasets are of varying difficulty with the MATH dataset catering to competition-level questions and GSM8K on the other hand handling only fundamental level mathematical questions. The nature of problems within these datasets exhibits variation. Common datasets typically contain mathematical word problems, whereas TAL-SCQ5K features multiple-choice questions. No mid-difficulty level dataset is available as a standard that has shown to perform equally on the different benchmarks mentioned. Systematic math datasets covering basic to high-level mathematics useful for undergraduate to graduate level students are still a limitation.

2.2 Large Language Models for Cryptography

Many LLMs were extensively evaluated on their mathematics reasoning capabilities based on the datasets mentioned in the previous sections. As the root concepts of cryptography are based on foundational mathematics, it is implied that the mathematical capability of a model will directly impact the model's cryptographic reasoning capability.

As of now, there are no publicly available cryptography datasets that can be used for LLM training. The limited evaluation done in this domain is mostly based on the Capture-the-Flag challenges [25] which are publicly available on the competition websites and the university cryptography examination questionnaires [26]. However, there are no benchmark datasets available yet for training or extensive evaluation.

An interesting direction about the limited capability of LLM to perform basic decryption (which is at the core of cryptography) is studied by Yuan et. al [34]. They state that

the SOTA LLMs can bypass the safety protocols when input is encrypted, which raises a big question about LLM’s struggles in this domain and its impact on AI Security.

Noever [24] also discusses the capability of GPT4 in understanding ciphers. This paper studies various low to medium-difficulty ciphers (mostly ones concerning letter ordering) on their *CipherDataset* with 654 test cases and found that only the lower-difficulty ciphers like Caesar cipher, Morse Codes, and LeetSpeak could be decrypted with GPT4. The drawback of the study was the size of the dataset which had only 13 test samples, which lacked the breadth and complexity to represent the full spectrum of these encryption methods.

2.3 Data Augmentation Techniques

Multiple studies have shown that data augmentation techniques have a transformative impact on LLM’s understandability. LLMs can also facilitate the creation of synthetic datasets of high quality sometimes exceeding the value of data curated by humans [35]. This helps to address the challenge posed by the limited supply of human-annotated data, which is much more expensive.

Yu et al., [17] in their study, showed that they could augment 15k samples to 300k with question bootstrapping. In this process, they used techniques like rephrasing using LLM system prompt, FOBAR method [36], and self-verification methods [37] to augment the original GSM8K [28] and the MATH [29]. This augmented dataset was eventually used to fine-tune a newer model, which showed an improvement in performance in GSM8K and MATH by 11.55% and 8.7% respectively.

Dai and Adel [38] also studied data augmentation techniques on named entity recognition tasks and found that even the simple data augmentation techniques can improve over the baseline where no augmentation is used. However, no specific augmentation method was seen to do well for RNNs and transformers. Another approach, used by Zhang et al. [39] and Wei and Zou [40], involves replacing words with synonyms found in a digital reference like WordNet, a large English language thesaurus. Wei and Zou also randomly swapped two words in a sentence to augment text classification training sets.

Some direct data rewriting tasks were also explored by Dai et al., [41] who propose a new augmentation approach AugGPT that rephrases each sample into multiple conceptually similar but semantically different samples.

2.4 Fine Tuning

There have been many fine-tuning approaches that are used extensively to adapt the open-source LLMs to downstream tasks. The popularity of fine-tuning in LLMs is specifically attributed to the research on Parameter Efficient Fine Tuning (PEFT) methods like Adapters [42], LoRA [4], QLoRA [43] etc. There have been multiple research to study the impact of both full parameter fine-tuning and PEFT from different dimensions. For example, Wang et al. [44] studied the impact of input format prompts on the generalization of the fine-tuning outputs. Baldazzi et al. [45], on the other hand, also made a successful attempt to achieve domain-specific fine-tuning using Enterprise Knowledge Graphs.

Like the other tasks, there have also been several attempts to fine-tune LLMs to improve their mathematical reasoning capabilities. Yang et al. [46] conducted a research where they showed close to perfect accuracy with multi-digit arithmetic problems (eight+ digits) with their fine-tuned model MathGLM. MathGLM is a 2 billion parameter model, trained on the GLM-Dialog model [47]. WizardMath [48], a fine-tuned model on Llama 2 has also enhanced the mathematical reasoning abilities of the base model, by applying Reinforcement Learning from the Evol-Instruct Feedback (RLEIF) method to the domain

of math. Around the same time, MAMmoTH [49] models are trained on the MathInstruct dataset curated from 13 math datasets and used Chain of Thought (CoT) and Program-of-Thought (PoT) rationales for teaching LLMs to use external tools like the python interpreter for solving mathematical problems. MetaMATH [17] also achieved improved results on GSM8K and MATH datasets with their augmented dataset MetaMATHQA.

Some studies like by Soygazi and Aguz [50] also investigated the role of LLMs and LangChain in mathematics education in their work and gave a descriptive analysis of how chained LLMs (LLMMathChain) provide many accurate results compared to ChatGPT, but the accuracy of both in mathematical domains are below expectations.

Our intended dataset will also try to achieve a good qualitative performance in the domain of cryptography on a fine-tuned model. Overall, the LLM capabilities in cryptographic tasks remain unexplored in the literature. However, knowledge transfer from related fields (e.g., mathematics) is a potential first step for cryptographic primitives.

Chapter 3

Methodology

The following section talks about the methodology followed during the study. The section is divided into two major parts: **Dataset Preparation** (section 3.1) and **Fine Tuning** (section 3.2) process used during the design of the experiment.

3.1 Dataset Preparation

To facilitate the fine-tuning of the LLMs for the domain of *cryptography*, a meticulously curated dataset specifically designed for this purpose was developed, namely **CryptoQA**. This process involved several crucial data augmentation and annotation techniques to enrich the information and enhance the LLM's learning potential. Data collection and processing formed the backbone of this pipeline, to ensure the quality and consistency of the incorporated information. The following section delves into this multi-stage pipeline, outlining the various steps undertaken to construct the robust CryptoQA dataset.

3.1.1 Task and Scope of the Dataset

This section details the *specific objectives*, *target users*, and *content* of the curated cryptography dataset.

- **Target users:** Recognizing the extensive nature of cryptography, CryptoQA primarily caters to individuals seeking to establish a conceptual understanding of the field's core principles and practical applications. The scope will be increased further as the size of the dataset increases.
- **Complexity Level:** The dataset prioritizes conceptual information and cipher problems of easy to medium in difficulty. This targeted focus, owing to the small dataset, ensures accessibility for those new to the field while providing valuable examples for those seeking to solidify their grasp of cryptographic concepts.

Additionally among the cipher problems only “encryption” and “decryption” problems are focused for now, where we have a fixed way/algorithm of solving the problems. CryptoQA dataset currently has a ratio of approximately **5:1** of conceptual cipher questions to encryption-decryption cipher problems. The rest of the dataset consists of basic math problems specifically required for solving cryptographic tasks.

Dataset samples involving using a CoT approach for cracking an encrypted text, without specifying the key are *not* attempted as part of this preliminary research.

- **Content Breakdown:**

The dataset delves into various sub-fields of cryptography with a particular emphasis on the following key areas:

- **Number Theory:** This fundamental branch of mathematics provides the essential foundation for numerous cryptographic algorithms. The dataset will explore relevant *number theory* concepts that are critically important for comprehending cryptographic operations. The topics covered are *Long Division*, *Divisibility Criteria*, *GCD and LCM*, *Extended Euclidean Algorithm*, and *Prime Numbers*.
- **Probability:** Counting Theorem, Permutations, and Combination problems are covered.
- **Classical Encryption Techniques:** It includes a conceptual understanding of symmetric cipher models, substitution techniques, and transposition techniques.
- **Block Ciphers:** The topics covered are conceptual understanding of block structures and their principles and Data encryption standards (DES) with examples.
- **Finite Fields:** Definitions and principles of groups, rings, fields, and finite fields, are covered with simple examples.

3.1.2 Data Collection and Source

This dataset leverages the power of question-answer pairs, encompassing both numerical problems for practical application and conceptual inquiries to solidify foundational understanding. Popular academic texts are selected for this purpose. The selection process rigorously prioritizes academic integrity, guaranteeing that the chosen texts represent high-quality resources. This emphasis on quality is particularly crucial in the early stages of our research, where the overall dataset size is still relatively limited.

The chosen academic texts are as follows :

- Cryptography and Network Security - Principles and Practice, William Stallings [51]: Eighth Edition
 - Chapter 3 - Classical Encryption Techniques (Page 21 to 40)
 - Chapter 4 - Block Ciphers and Data Encryption Standards (Page 112 to 130)
 - Chapter 5 - Finite Fields (Page 135 to 150)
- A first course in Probability, Sheldon Ross [52]: 10th Edition
 - Chapter 1 - Combinatorial Analysis (Page 13 to 24)
 - Chapter 2 - Axioms of Probability (Page 34 to 47)
- A Gentle introduction to number theory and cryptography, Luis Finotti [53]
 - Important Sets
 - Long Division
 - A Useful Theorem and Some Semantics
 - Simple Divisibility Criteria
 - GCD and LCM
 - The Extended Euclidean Theorem
 - Prime Numbers

Another requirement of the research was an analysis of open-source math datasets for the first stage of the fine-tuning process. More on this will be covered in the fine-tuning (section 3.2) section. The datasets used for the analysis were as follows:

- **Mathematics Aptitude Test of Heuristics (MATH)** [15] - The Mathematics Aptitude Test of Heuristics (MATH) dataset consists of problems from **mathematics competitions**, and consists of step-by-step solution written in LaTeX and natural language for better interpretability. The step-by-step solution contains the final answer enclosed in LaTeX's \boxed tag.
- **GSM8K** [19] - GSM8K (Grade School Math 8K) is a dataset of 8.5K high quality diverse **grade school math word problems**, created to support the task of question answering on basic mathematical problems that require multi-step reasoning. The reasoning dataset has 2-8 steps and is much less challenging than the MATH dataset. The dataset is entirely in natural language.
- **MetaMATHQA** [17] - MetaMATHQA is a mathematical dataset that is bootstrapped mathematical questions and rewritten from multiple perspectives, providing a much larger **augmented dataset** obtained from the training samples of [19] and [15].

3.1.3 Cleaning and Preprocessing

This section elaborates on the characteristics and processing of the dataset to be constructed for LLM’s training in the domain of cryptography. The core structure of the dataset uses question-answer pairs. This format aligns effectively with the LLM’s training paradigm, facilitating the model’s ability to learn relationships between inquiries and corresponding responses within the cryptographic domain. To further enrich the dataset and promote reusability for broader research endeavors, each question-answer pair is augmented with additional contextual information which will be covered more in section 3.1.5.

This standardization pipeline takes the following steps:

Tokenization

Token is the basic unit of a LLM. Tokenization is the process of translating strings of text into smaller units, like a *character*, a *part of a word*, the *whole word*, or sometimes even a larger one like a *whole phrase*.

In the context of LLMs, tokenization is considered a completely separate stage of the pipeline which is executed as a preprocessing stage at the beginning of the LLM training. This stage includes a different training stage with a separate dataset which is generally different from the LLM pretraining dataset. An illustration of how a tokenizer model is used to obtain the token sequence to feed it to the LLM is shown in Figure 3.1.

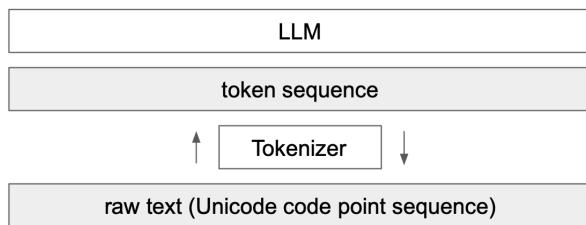


Figure 3.1: Illustration of Tokenization in LLM [3]

Tokenization is a critical part of the LLM training process, allowing LLMs to tap into their creative potential. It also influences the occasional oddity and the use of prompt manipulation techniques seen in widely available LLMs. Some common tokenization algorithms include Byte Pair Encoding (BPE), WordPiece, and SentencePiece.

The state-of-the-art LLMs usually use tokenizers that tokenize text into larger chunks, or “*subwords*”, instead of the basic character level or word level tokens. This approach of using *chunks of words* instead of characters significantly decreases the token count for a sequence of text and thus allows a much wider context window that will contain much more information than the previous case. However, increasing the number of tokens to a very big number also brings along a penalty - as we increase the number of tokens, the embedding table grows larger and larger. Eventually, the size of the output layer the transformer architecture which uses softmax to do the final prediction of the next token, also grows.

The current state-of-the-art models usually have a vocabulary size within the range of 30k to 100k. Open-source LLMs like the Llama and Mistral series use the llama tokenizer which has a vocabulary size of 32k and closed LLMs like GPT-2 and GPT-4 have a size of 50k and 100k respectively.

As our research is concerned with only open source small-scale LLMs we use the *Llama tokenizer* to tokenize our input samples as recommended in the official Llama model documentation.

An experimentation script to analyze the tokenizer we use in our project can be found in `model-preliminary-tests/llama_tokenizer/llama2_tokenizer_check.ipynb` in the codebase which is publicly available.

Standardization

This stage focuses on ensuring consistency within the dataset. It encompasses two primary aspects:

- **Numerical Representation:** In this step, numerical expressions within the dataset are converted into a standardized format, such as the common *LATEX* standard for mathematical notation, wherever necessary. This particularly applies to the math problems in the datasets with numerical inputs. This promotes uniformity and facilitates the LLM’s understanding of these expressions [15].

Need for consistent numerical representation: This standardization step allows us to flexibly encode mathematical problems while avoiding unusual symbols or cumbersome formal languages. Also, the domain of cryptography demands a lot of figures and graphs for a better understanding of cryptographic algorithms. Thus, the use of *LATEX* formatting allows the language models to process figures, diagrams, and graphics, making it possible to encode the context or the data from a graph or table to some extent for feeding it to the language model as a sequence of characters.

- **Chain of thought (CoT) approach** [54]: This emerging technique, as outlined in the work by Wei et al., incorporates an explicit CoT alongside the question-answer pairs. This CoT can be textual or symbolic and represents the reasoning steps that lead to the final answer. Including the CoT information in the LLM training sample, the training process has the potential to enhance the ability of the LLM not only to provide correct answers but also to explain the reasoning behind those answers during generation. This approach allows the model to decompose an answer to multiple intermediate reasoning steps and suggests how it might have arrived at the final answer, providing opportunities to debug where the reasoning path went wrong.

The CryptoQA dataset consists of a combination of both word and numerical problems. We have tried to write detailed and well-explained answers, instead of direct one-word or keyword-based short answers, adhering to the CoT adopted for *numer-*

ical solutions. Figure 3.2 shows a sample from the CryptoQA dataset with and without CoT.

Chain of Thought vs Direct Answer
Sample with Chain of Thought Question: How many different 7-place license plates are possible if the first 3 places are to be occupied by letters and the final 4 by numbers? Answer: By considering the choice of a license plate as the outcome of seven separate experiments: three experiments for choosing letters and four experiments for choosing numbers. Thus, the total number of possible license plates is $26 \times 26 \times 26 \times 10 \times 10 \times 10 \times 10 = 175,760,000$.
Sample without Chain of Thought Question: How many different ordered arrangements of the letters a, b, and c are possible? Answer: 6

Figure 3.2: Chain of Thought approach used in CryptoQA

For our dataset, we are using conventional natural language CoT, from the academic training data available (Section 3.1.2). There are also other approaches to designing CoT answers as well, like using the program CoT’s [55].

We use few-shot prompting on the `gpt-3.5-turbo` get the CoT answers from the academic texts. More about few-shot prompting will be discussed in the next section on In-Context Learning (3.1.3).

In-context Learning

In-context Learning (ICL) is a method of prompt engineering, where we give examples of the tasks in the prompt to narrow down the solution space of the generation. Adhering to the standardization and format of the dataset necessary for LLM’s training, ICL is used to create the dataset. In this method, we can have one-shot or few-shot prompting [56] to teach the model to generate the particular format of response we are interested in. In this method, there is no change in the weights of the model. For our study, we have used few-shot prompting to generate the samples for our CryptoQA.

Our final aim is to have a *better zero-shot accuracy* for the trained model.

Context length is an important parameter of the LLMs, which influences the results of the LLMs. Context length is the number of tokens a language model can process at once. For example, llama 2 has a context size of 4k whereas, large models like GPT 4 have a context size of 32k. It is analogous to the *memory* or *attention span* of the model. Although it might seem that increasing the context length will linearly increase the memory of the language models and always give better results, it might be so for some cases but Chen et al. [57] shows that the accuracy doesn’t grow linearly. Also, increasing the context length largely increases the computational costs.

We use `gpt-3.5-turbo` with a context length of 4k, to generate the dataset samples in the required format.

Our task required dataset reformatting of the large training text available from the academic texts. We used two-shot prompting for this task to get the samples. Since this task only involved creative rearrangement of the training data and no real computations where accuracy could have been a concern, this technique was found to be reliable.

If there is an attempt to generate new data samples from scratch where answers are unknown and not indicated in the input prompt, we might require up to 50-shot prompting, depending on the accuracy of the tasks.

We used four types of two-shot prompts for rearranging the metadata and generating the augmented samples as shown in Appendix section B.

- **System Prompt:** This is the prompt used to generate the original question-answer pairs from the training data. See Figure 5 in the Appendix section.
- **Rephrasing Prompt:** This is the prompt used to generate the `rephrased` version of the original sample for the augmented dataset (Section 3.1.5). See Figure 6 in the Appendix section.
- **Value Modification Prompt:** This is the prompt used to generate the `value modified` version of the original sample for the augmented dataset (Section 3.1.5). See Figure 7 in the Appendix section.
- **Backward Reasoning Prompt:** This is the prompt used to generate the `backward reasoning` version of the original sample for the augmented dataset (Section 3.1.5). See Figure 7 in the Appendix section.

Role Playing

In addition to using ICL to generate the data for the dataset, another technique applied to channel the data for generating controlled responses is role-playing in LLMs.

In this technique, the language model is forced to play a certain role such that it does not deviate from the expected solution/response. There has been a lot of research around this domain of investigating which social roles can help generate the best response [58]. This technique is also a defensive measure against prompt hacking, where malicious inputs can generate offensive statements in the response. More about how this technique is used for dataset generation in the prompts section (Section B).

Monolingual Data

To ensure consistency in language processing, the current approach focuses on using English-language data only. This simplifies the pre-processing steps and avoids potential complexities associated with multilingual datasets.

3.1.4 Data Annotation

The preprocessed data is further enriched with informative labels. These labels create a structured format that helps prepare a uniform and balanced dataset for fine-tuning.

- **Labels:** Each data sample is assigned a set of labels to ensure a good train-test split, including:
 - `question`: The question or problem statement of the dataset.
 - `answer`: The answer to the question in the previous field. The answer is strictly written using the CoT format, wherever possible.
 - `type`: This label identifies whether the information within the question or answer pertains primarily to a text problem (`word`), numerical problem (`math`), or cipher problems containing mathematical computations (`cipher-math`).
 - `category`: This label differentiates between original question-answer pair (`orig`) and sample generated by augmentation i.e. (`valmod`, `rephr` and `bkwrd`). More on the augmentation techniques used and the meaning of the categories specified will be covered in the section 3.1.5.

- **topic:** This label specifies the specific area of cryptography being addressed, such as **probability**, **numbertheory**, and **cipher**. This topical categorization allows the LLM to focus on relevant domain knowledge when processing the information.
- **source:** Initials of the academic texts or sources used. This category does not pose any additional meaning other than metadata of the dataset creators.
- **Annotation Techniques:** This approach involves crafting specific prompts that guide **gpt-3.5-turbo** in identifying and labeling key aspects of the data. This manual annotation process, particularly in the initial stages of dataset preparation, aims to ensure the accuracy and consistency of the labels, ultimately contributing to a robust and informative dataset for LLM training. However, this step still requires manually confirming the sample generated by **gpt-3.5-turbo**.

3.1.5 Data Augmentation

Data Augmentation is a process where we artificially generate new data from the already available training data to increase the size of the dataset. This is different from generating *synthetic data* where we artificially generate data without using training data.

There are several text-based data augmentation techniques like *word or sentence shuffling*, *rephrasing*, etc. In this study, we additionally use a newer technique mostly used for mathematical datasets called *backward reasoning* and other traditional input-based augmentations as discussed below:

- **Backward reasoning**

Backward Reasoning is an approach where a mathematical reasoning question is asked backward i.e. given the answer with a given condition we think backward to determine an unknown variable in the question. The unknown variable is usually masked with a variable 'x'. This type of augmentation to the original question is possible by two methods: FOBAR [36] and Self Verification [37]. The backward reasoning is considered particularly complicated to interpret than the corresponding forward task [59, 60] for humans, and the same is assumed for LLMs [61]. A dataset sample of CryptoQA for depicting forward reasoning questions are shown in Figure 3.3.

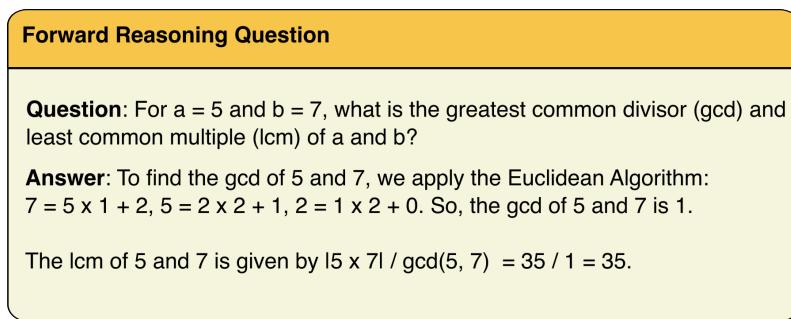


Figure 3.3: Forward reasoning sample from CryptoQA

In the self-verification technique the question with the answer (in the original question) is first rewritten into a declarative statement, e.g., “How many pages did he write?” (with the answer 50) is rewritten into “He wrote 50 pages”. Then, a question for asking the value of x is appended to the question. A sample depicting backward reasoning using “Self verification” is shown in Figure 3.4.

Backward Reasoning Question- Self Verification

Question: For $a = 5$ and $b = y$, the greatest common divisor(gcd) is 1 and the least common multiple (lcm) is 35. What is the value of y ?

Answer: To solve for x given that the gcd of 5 and y is 1 and the lcm is 35, we can use the relationship between the gcd and lcm of two numbers. The formula for the lcm of two numbers a and b is given by $|a \times b| / \text{gcd}(a, b)$.

Given that the gcd of 5 and y is 1, we can substitute these values into the formula for the lcm:

$$\text{lcm}(5, y) = (5 \times y) / \text{gcd}(5, y)$$

Since the gcd is (1), the formula simplifies to:

$$\text{lcm}(5, y) = 5y$$

We are told that the lcm is 35, so we set $5y = 35$ and solve for y :

$$5y = 35 ; y = 35/5 ; y = 7$$

Therefore, the value of y that satisfies the given conditions is 7.

Figure 3.4: Backward reasoning sample (Self Verification Question) from CryptoQA

Whereas for the FOBAR technique, we directly append the answer to the question, i.e., “If we know the answer to the above question is 50(say), what is the value of unknown variable x ? An illustration of the FOBAR technique used in CryptoQA is shown in Figure 3.5.

Backward Reasoning Question - FOBAR

Question: For $a = 5$ and $b = y$ and greatest common divisor : $\text{gcd}(5,y)$ is 1, what is value of the least common multiple (lcm)? *If we know the answer to the above question is 110, what is the value of unknown variable x?*

Answer: To solve for x , given that the gcd of 5 and y is 1 and the lcm is 35, we can use the relationship between the gcd and lcm of two numbers. The formula for the lcm of two numbers a and b is given by $|a \times b| / \text{gcd}(a, b)$.

Given that the gcd of 5 and y is 1, we can substitute these values into the formula for the lcm:

$$\text{lcm}(5, y) = (5 \times y) / \text{gcd}(5, y)$$

Since the gcd is (1), the formula simplifies to:

$$\text{lcm}(5, y) = 5y$$

We are told that the lcm is 35, so we set $5y = 35$ and solve for y :

$$5y = 35 ; y = 35/5 ; y = 7$$

Therefore, the value of y that satisfies the given conditions is 7.

Figure 3.5: Backward reasoning sample (FOBAR question) from CryptoQA

We have a set of 314 samples of original questions and we augment the dataset samples to increase the dataset interpretability by the LLM. In the CryptoQA dataset, these samples are under the category column with the label `bkwrd`. This step is only possible for a particular structure of numerical questions of the dataset.

• Input Space Augmentation

Input space augmentation enriches the dataset by strategically manipulating the numerical parts of the existing question-answer pairs. It serves as a valuable tool for enriching the dataset and empowering the LLM to develop a deeper and more generalizable understanding of cryptographic principles, enhancing the model's ro-

bustness.

Several methods are implemented in input space augmentation within the context of our cryptographic dataset:

- **Value Modification:** This strategy focuses on modifying the numerical values present within the questions. For instance, a question originally asking about the modular inverse of a specific number could be transformed into questions with different numerical values while maintaining the underlying mathematical operation as shown in Figure 3.6. In the CryptoQA dataset, these samples are under the `category` column with the label `valmod`.

We get the samples for value modification augmentation using the “value modification prompt” used in the `gpt3.5-turbo` API. The prompt used is shown in Figure 7 in the Appendix section.

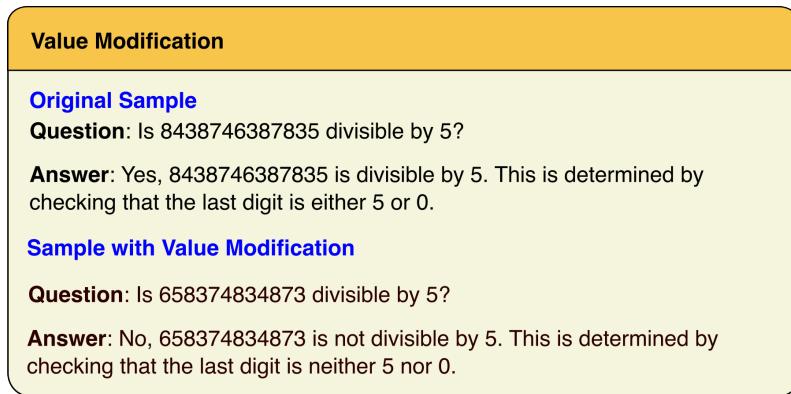


Figure 3.6: Value Modification in CryptoQA

- **Paraphrasing and Rephrasing:** This strategy involves rephrasing the existing questions while preserving their meaning. This can be achieved through *synonym substitution*, *sentence structure alteration*, or even the use of *different grammatical forms* as shown in Figure 3.7. In the CryptoQA dataset, these samples are under the `category` column with the label `rephr`.

We additionally use LLM rephrasing using few-shot prompt techniques by giving the original question and obtaining the rephrased questions. The rephrasing prompt is given in Figure 6 in the Appendix section.

3.1.6 Data Format Conversion

Following the data augmentation stage, the pre-processed dataset undergoes a final conversion step to ensure optimal compatibility with the LLM’s training framework. This conversion process involves transforming the data into a format that aligns with the specific requirements of the chosen LLM configuration.

The two standardized training formats used for the dataset used in this project are:

- Llama chat prompt template:

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

{{ user_message }} [/INST]
```

Rephrasing

Original Sample

Question: What are the different types of encryption systems based on the key usage between sender and receiver?

Answer: Encryption systems are classified based on key usage between sender and receiver. If both parties use the same key, it's known as symmetric, single-key, secret-key, or conventional encryption. Conversely, if the sender and receiver use different keys, the system is referred to as asymmetric, two key, or public-key encryption.

Sample with Rephrasing

Question: How are encryption systems categorized based on the key sharing between the sender and receiver?

Answer: Encryption systems are differentiated by the key sharing method between the sender and receiver. When both parties share the same key, the system is categorized as symmetric, single-key, secret-key, or conventional encryption. On the other hand, if the sender and receiver each use a unique key, the system is identified as asymmetric, two-key, or public-key encryption.

Figure 3.7: Rephrasing in CryptoQA

- Alpaca Style Format:

```
### System:  
{system_prompt}  
  
### User:  
{user_prompt}  
  
### Assistant:
```

Employing this standardized format offers several advantages:

- **Optimized for Performance:** Official prompting methods (as provided by the pre-trained LLM provider) are designed to work seamlessly with the underlying architecture of the model. Adhering to the format leads to the most effective responses from the model.
- **Compatibility:** The format ensures compatibility with the chosen LLM system, streamlining the integration of the dataset into the training pipeline.

It is important to acknowledge that other prompt formatting styles exist, such as ChatML. These styles might offer different functionalities or suit specific LLM systems. The selection of the optimal format depends on the chosen LLM framework and the desired training objectives.

By implementing this data format conversion step, the preprocessed dataset is transformed into a format readily consumable by the LLM training system.

3.1.7 CryptoQA Dataset

Considering all the major steps mentioned above, we construct the CryptoQA dataset. The CryptoQA dataset consists of 510 training and 50 testing question-and-answer pairs of the cryptographic domain, designed to assess an LLM's understanding of cryptography.

Figure 3.8 shows how one sample of a dataset looks like.

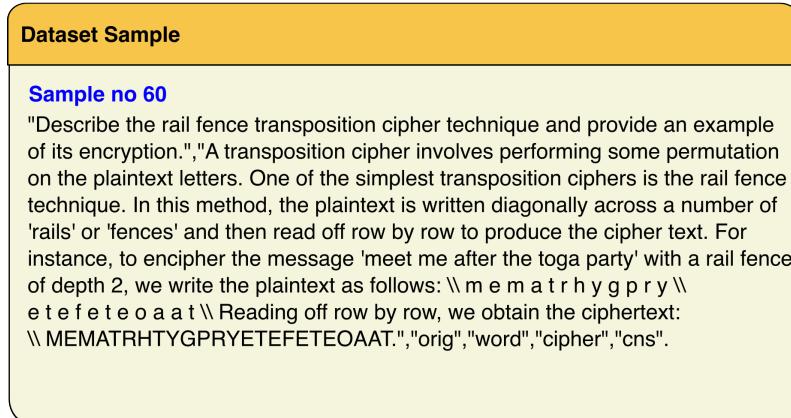


Figure 3.8: Sample of a dataset in CryptoQA

Dataset Information

- Train data samples: 510
- Test data samples: 50
- Dataset format: Comma-separated file (CSV)
- Dataset Link: [cryptoqa-v1](#)
- Using the dataset

```
from datasets import load_dataset
dataset = load_dataset("Manpa/cryptoqa-v1")
train_dataset = load_dataset("Manpa/cryptoqa-v1", split="train")
```

The distribution of the data examples according to the categories is shown in Figure 3.9, 3.10, 3.11.

You can also find additional examples for different value type of the problem in the Appendix section A.

3.2 Fine Tuning

3.2.1 Techniques used

The following techniques are used during the fine-tuning process with the CryptoQA dataset.

Percentage distribution of type in the dataset

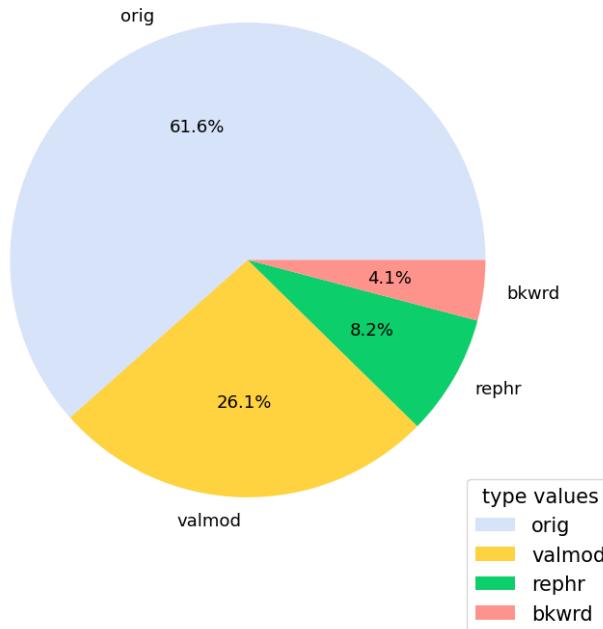


Figure 3.9: Percentage Distribution of types in CryptoQA

Percentage distribution of category in the dataset

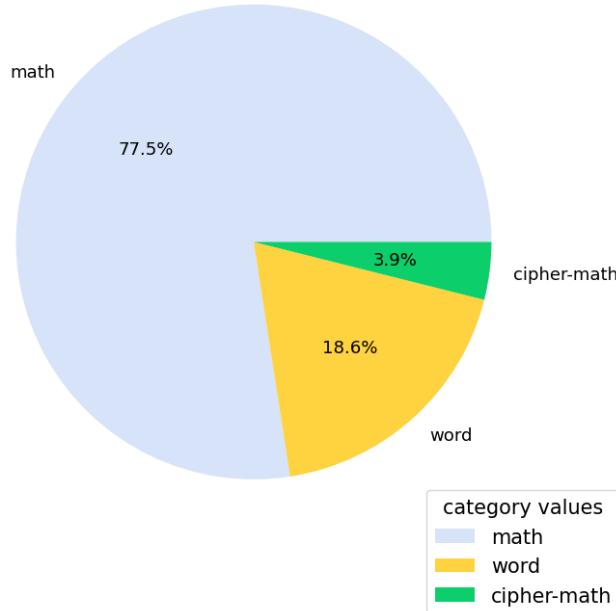


Figure 3.10: Distribution of categories in the CryptoQA dataset

Supervised Fine Tuning

The two most prominent fine-tuning approaches of LLMs include Supervised Fine Tuning (SFT) and Reinforcement Learning from Human Feedback (RLHF). Both contribute towards training a model to predict the next best possible word or move but greatly vary in their approaches to doing so.

Supervised Fine Tuning takes place through carefully curated high-quality labeled

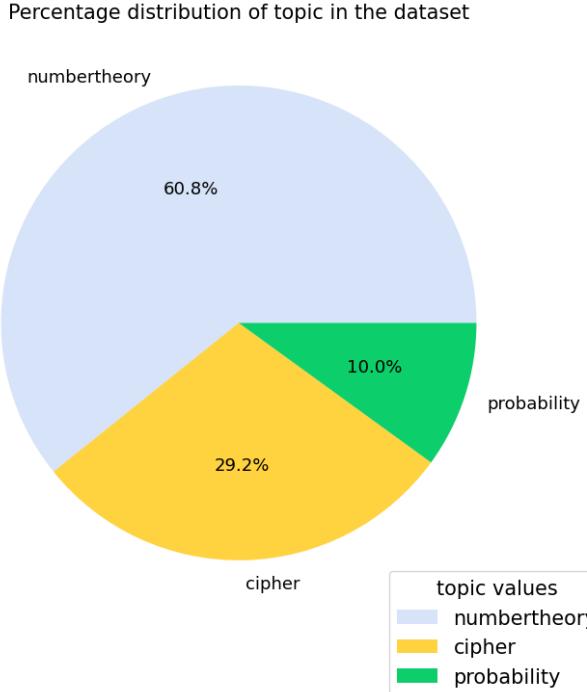


Figure 3.11: Percentage Distribution of topics in CryptoQA

examples and then fine-tuning the transformer-based LLMs with multiple forward and backward passes. This method is particularly effective for tasks where we have fixed answers to questions.

Whereas in **Reinforcement Learning from Human Feedback** approach, the model learns by interacting with its environment and receiving feedback. In this type of training, we need no data i.e. the model doesn't learn from a static dataset, but from real-world experiences and human-guided rewards or penalties. This approach is particularly beneficial for tasks where external explicit labeling is difficult, for example for tasks involving human opinion or human values. Although RLHF is advantageous, it also has drawbacks like requiring human involvement for providing evaluations and rewards and training a separate reward model.

In this project, we use SFT as it is our task demands a structured dataset for training, ideal for problems with clear answers. We also can improve the accuracy of this task by increasing the quality and quantity of the dataset, improving the SFT results.

Parameter Efficient Fine Tuning (PEFT)

Parameter Efficient Fine Tuning [62, 63] is a technique where we update a subset of parameters of the pre-trained LLMs. Fundamentally this method is different from full fine-tuning as in full fine-tuning usually all the parameters of the models are updated, whereas in PEFT we only update a *subset* of it and keep the rest of the parameters frozen i.e reducing the number of trainable parameters.

For LLMs, using the full parameter training method is quite rare as the model sizes are in the range of billions which will require quite high computations. For every adaptation to a new task, the fine-tuning expenses become a bottleneck. Smaller models can be used for production settings, however, multiple smaller models will be required which will also not generalize well keeping in view the current commercial LLM models.

For consumer settings i.e.in small to medium-scale industries and academia, even time is a big factor which even increases further with limited resources. For industries expecting

to deploy these models, full parameter fine-tuning becomes even more complicated to serve and handle, online training is a requirement to keep up with the latest data and it's almost impossible to train specialized models for different users, owing to its time and money requirement. Although it might be theoretically possible to retrain the model after each new user's input, it will be overly complicated to handle, in addition to the massive storage requirement.

Therefore, to democratize the process of fine-tuning we use the PEFT methods. There are many PEFT methods in use today like *Adapters* [64], *Low Rank Adaptation (LoRA)* [4], *Prefix tuning* [65], *Prompt tuning* [66], *P-tuning* [67], *Infused Adapter (IA3)* [68], etc. These methods are largely used as they significantly reduce the computational requirements and also don't underperform significantly.

We use the Low-Rank Adaptation technique in our fine-tuning as it has shown significantly improved performance than the other PEFT methods like Adaptors and empirically performs well in most cases.

Low Rank Adaptation(LoRA)

LoRA [4] is a PEFT method that is based on the principle that there exists a low-dimensional re-parameterization space that is as effective as the full over-parameterized space of the LLMs. Another interesting conclusion from this paper was that the larger the foundational models, the lower the intrinsic dimension i.e. the fine tuning can be done on much fewer parameters. Following the findings, the main idea behind LoRA is that the change in weights during model training also has a lower intrinsic dimension.

We know that the weight matrix update in each training step is given by

$$h = W + \Delta W = W + BA$$

In LoRA fine tuning we decompose the ΔW matrix using rank decomposition, where if ΔW is of dimension $n * k$, it can be expressed as the product of two low-rank matrices B_{n*r} and A_{r*k} where $r \ll \min(n, k)$ is the rank of the LoRA.

In Figure 3.12, suppose x is the input and W is the original weight matrix, then in the forward pass of the layer, the original Wx is modified to $Wx + BAx$. A random Gaussian initialization is used for A and B is initialized to 0, such that $BA = 0$ at the start of training. It is typically applied to the attention weights in a transformer.

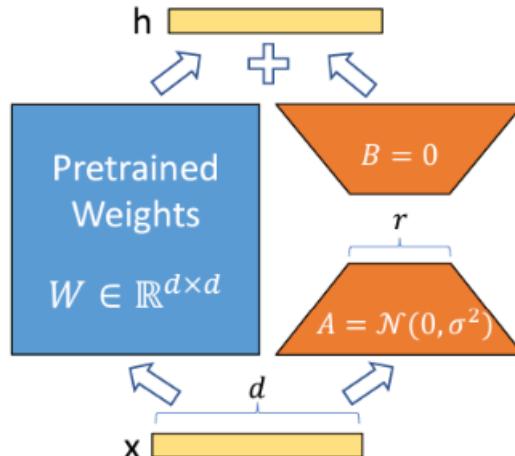


Figure 3.12: LoRA reparameterization; Only A and B matrices are trained [4]

Number of trainable parameters - CryptoLLM			
Model Name	Full Training Parameters	LoRA Training Parameter	LoRA Training Parameter %
crypto-llama7b-math	6771970048	33554432	0.4954
crypto-llama7b	6771970048	33554432	0.4954
crypto-mistral7b-metamath	7279480832	37748736	0.5185
crypto-metamath	3789828096	37748736	0.996

Table 3.1: The number of trainable parameters and its percentage compared to the full parameter space using LoRA

How useful is LoRA?

- In practice, it drastically reduces the number of parameters as compared to the full fine-tuning. For instance, according to the paper LoRA can reduce the number of trainable parameters by 10,000 times (as shown in Table 3.1) and the GPU memory requirement by 3 times. [More about the versions of the LLM's trained will be covered in the section 3.2.3]
- The LoRA weights are merged with the original model weights during inference without additional computational overhead i.e. the inference time remains the same.

Quantized Low-Rank Adaptation (Q-LoRA)

QLoRA is a quantized form of LoRA that uses a special datatype called NormalFloat which allows a normal distribution of weights to be compressed from 16-bit float (compressed during LoRA) into 4-bit floats and then restores at the end with minimal loss in accuracy. So our final weights are 16-bit float precision.

This method also reduces the memory footprint, making it feasible to run LLM models even on consumer GPUs as shown in Table 3.2. Researchers in a study [69] of finding the optimal number of bits required to maintain the zero-shot accuracy, ran 35000 experiments and found that 4-bit quantization is the most optimal for the present SOTA LLMs.

Model	Original Size (FP16)	Quantized Size (INT4)
Llama2-7B	13.5 GB	3.9 GB
Llama2-13B	26.1 GB	7.3 GB
Llama2-70B	138 GB	40.7 GB

Table 3.2: This table illustrates the compute comparisons of different llama models with and without quantization (Adopted from tensorops.ai)

3.2.2 Choice of the Large language model for fine tuning

The LLMs selected for the analysis are **Llama-2 7B chat model** and **Mistral 7B parameter model**. These two models are specifically chosen for the following reasons:

- The open-source pre-trained Llama and Mistral series greatly simplify the process of fine-tuning. While the model's weights are available, information regarding the training data used to train these models remains undisclosed, leaving users without any details about the datasets used in obtaining the pre-trained models.
- There are proven results [70, 17] about these models being fine-tuned for mathematics tasks which offers the opportunity for working with mathematically heavy cryptographic models.
- Reasonable compute requirements for fine-tuning if we use parameter efficient fine-tuning and quantization methods as shown in Table 3.2. The number is in a similar range for the mistral series.
- Using smaller and open-source models also promotes reproducibility in the research community and democratizes the scope of incorporating such use cases into consumer-level devices.

3.2.3 Approach

As already stated in the previous sections, the study methodology takes place in two parts: Dataset Preparation (Section 3.1) and Fine Tuning (Section 3.2).

The fine-tuning pipeline is further divided into two sections:

- **Math fine-tuning:** In this step, we take a llama 7B chat and a base mistral 7B model and train it with MATH and MetaMATH datasets respectively to analyze its performance in both cases.

We also attempt to skip this step and directly train the CryptoLLMs on the already finetuned MetaMATH 7B parameter model for analyzing performance improvement in both cases.

- **Crypto fine-tuning:** In this step, we further fine-tune the model obtained in the previous stage with the CryptoQA dataset. We also analyze the impact of skipping the previous stage (math fine-tuning) on the results obtained.

The figure 3.13 below shows the flowchart of the working pipeline of the fine-tuning process:

There are four versions of the trained models:

- **crypto-llama7b-math:** In this version, we fine-tune the llama2 7B chat model on the MATH [29] at the first stage and further fine-tune the resulting model on the CryptoQA dataset.
- **crypto-llama7b:** In this version, we fine-tune the llama2 7B chat model directly on the CryptoQA dataset.
- **crypto-mistral7b-metamath:** In this version, we fine-tune the mistral 7b model on the MetaMATH dataset at the first stage and further fine-tune the resulting model on the CryptoQA dataset.

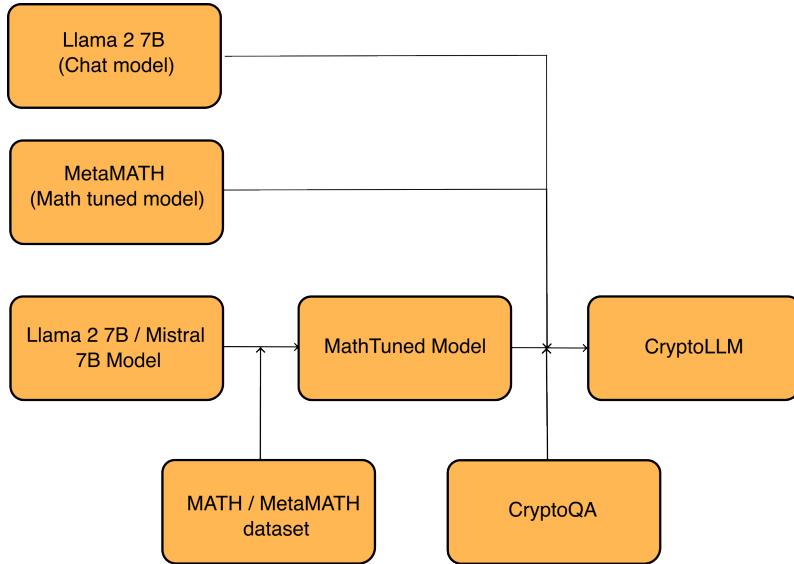


Figure 3.13: Fine Tuning Pipeline

- **crypto-metamath:** In this version, we already take a fine-tuned math model (MetaMATH model series) and fine-tune the crypto dataset over the math-tuned-metamath model.

The models were set and tested with the following configuration setup:

- **crypto-llama7b-math**

- base model stage 1: [llama-2-7b-chat model](#)
- fine-tuned model stage 1: llama-2-7b-math [[HF model card](#)]
- dataset trained in stage 1: MATH [29] by Hendrycks et al.
- dataset trained in stage 2: [CryptoQA](#)
- final fine-tuned model: crypto-llama7b-math [[HF model card](#)]
- training parameters:
 - * prompt format: Llama 2 official format as shown in section [3.1.6](#).
 - * tokenizer: Official Llama tokenizer + Padding token added as the end of sentence (eos) token (`<PAD>` = `<\s>`)
 - * LoRA parameters: `lora_r` = 64 ; `lora_alpha` = 16 and `lora_dropout` = 0.1
 - * 4-bit quantization used
 - * number of epochs: 10
 - * optimizer: `paged_adamw_32bit`
 - * scheduler: `cosine`
 - * learning rate: `2e-4`
 - * warmup ratio: 0.03
 - * weight decay: 0.001

- **crypto-llama7b**

- base model: [llama-2-7b-chat model](#)
- fine-tuned model: crypto-llama7b [[HF Model card](#)]

- dataset trained: [CryptoQA](#)
- training parameters:
 - * prompt format: Llama 2 official format as shown in section [3.1.6](#).
 - * tokenizer: Official Llama tokenizer + Padding token added as the eos token ($\text{<PAD>} = \text{<\$>}$)
 - * LoRA parameters: `lora_r = 64` ; `lora_alpha = 16` and `lora_dropout = 0.1`
 - * 4-bit quantization used
 - * number of epochs: 10
 - * optimizer: `paged_adamw_32bit`
 - * scheduler: `cosine`
 - * learning rate: `2e-4`
 - * warmup ratio: 0.03
 - * weight decay: 0.001

- **crypto-mistral7b-metamath**

- base model stage 1: [mistral 7B parameter model](#)
- finetuned model stage 1: mistral-7b-metamath [[HF model card](#)]
- dataset trained in stage 1: MetaMATHQA [17]
- dataset trained in stage 2: [CryptoQA](#)
- final finetuned model: crypto-mistral7b-metamath
- training parameters:
 - * prompt format: Alpaca format without system prompt as shown in section [3.1.6](#)
 - * tokenizer: Official Llama tokenizer + Padding token added as the eos token ($\text{<PAD>} = \text{<\$>}$)
 - * LoRA parameters: `lora_r = 64` ; `lora_alpha = 16` and `lora_dropout = 0.1`
 - * 4-bit quantization used
 - * number of epochs: 10
 - * optimizer: `paged_adamw_32bit`
 - * scheduler: `cosine`
 - * learning rate: `1e-05`
 - * warmup ratio: 0.03
 - * weight decay: 0.001

- **crypto-metamath**

- base model: [mistral 7B parameter model](#)
- finetuned model: crypto-metamath [[HF model card](#)]
- dataset trained: [CryptoQA](#)
- training parameters:
 - * prompt format: Alpaca format without system prompt as shown in section [3.1.6](#)
 - * tokenizer: Official Llama tokenizer + Padding token added as the eos token ($\text{<PAD>} = \text{<\$>}$)

- * LoRA parameters: `lora_r` = 64 ; `lora_alpha` = 16 and `lora_dropout` = 0.1
- * 4-bit quantization used
- * number of epochs: 10
- * optimizer: `paged_adamw_32bit`
- * scheduler: `cosine`
- * learning rate: `1e-05`
- * warmup ratio: 0.03
- * weight decay: 0.001

Chapter 4

Results and Discussion

4.1 Experimental Setup

4.1.1 Hardware Setup

Hardware for fine-tuning LLMs :

- **Graphical Processing Unit:** NVIDIA Tesla V100 SXM2 32 GB
- **Memory:** A minimum VRAM of 16GB for the 7B parameter models
- **Storage:** One checkpoint of the llama 13B parameter consumes about 5 gigabytes of storage.

4.1.2 Software Setup

- **Language:** Python 3.10+
- **Libraries required:** `torch`, `transformers`, `bitsandbytes`, `peft`, `trl`, `langchain`

4.2 LLM Training Performance

4.2.1 Loss Curves

In the SFT approach, the LLMs usually use Cross Entropy Loss to calculate the accuracy of the next prediction word. Cross entropy is a generalized form of the log loss that measures how “confident” the model is about its prediction based on the comparison with the true distribution.

A lower cross-entropy score indicates that the model’s predicted probability distribution is close to the true distribution. The llama and the mistral models calculate the loss values using this function.

In mathematical terms, the cross-entropy is defined for two discrete probability distributions, P (the true distribution of the data) and Q (the distribution predicted by the model), as:

$$H(P, Q) = - \sum_x P(x) \log Q(x)$$

Figure 4.1 shows the loss curves of the models fine-tuned on the mathematics datasets and figure 4.2 shows the learning rates for the llama and the mistral models used to train the models.

We further train the math-tuned models with the CryptoQA dataset and obtain the training loss characteristics as shown in Figure 4.3.

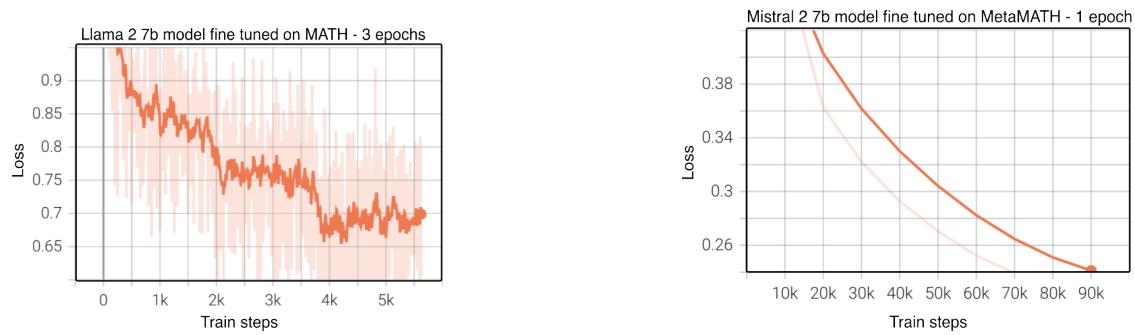


Figure 4.1: Loss Curve Characteristics of Llama and Mistral Mathematics tuned models

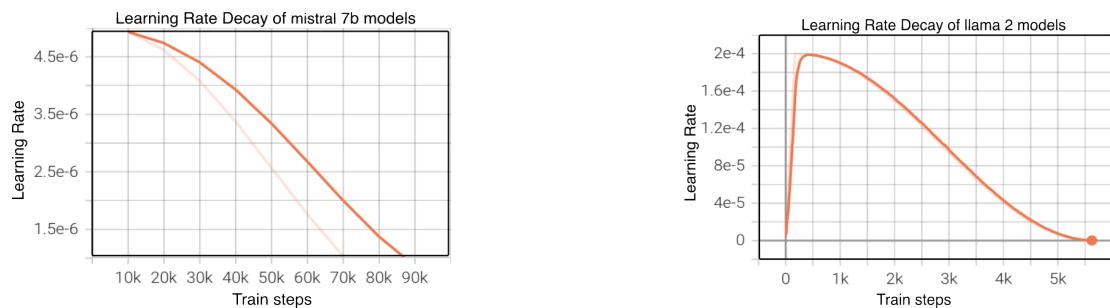


Figure 4.2: Learning Curve Characteristics of Llama and Mistral Mathematics tuned models

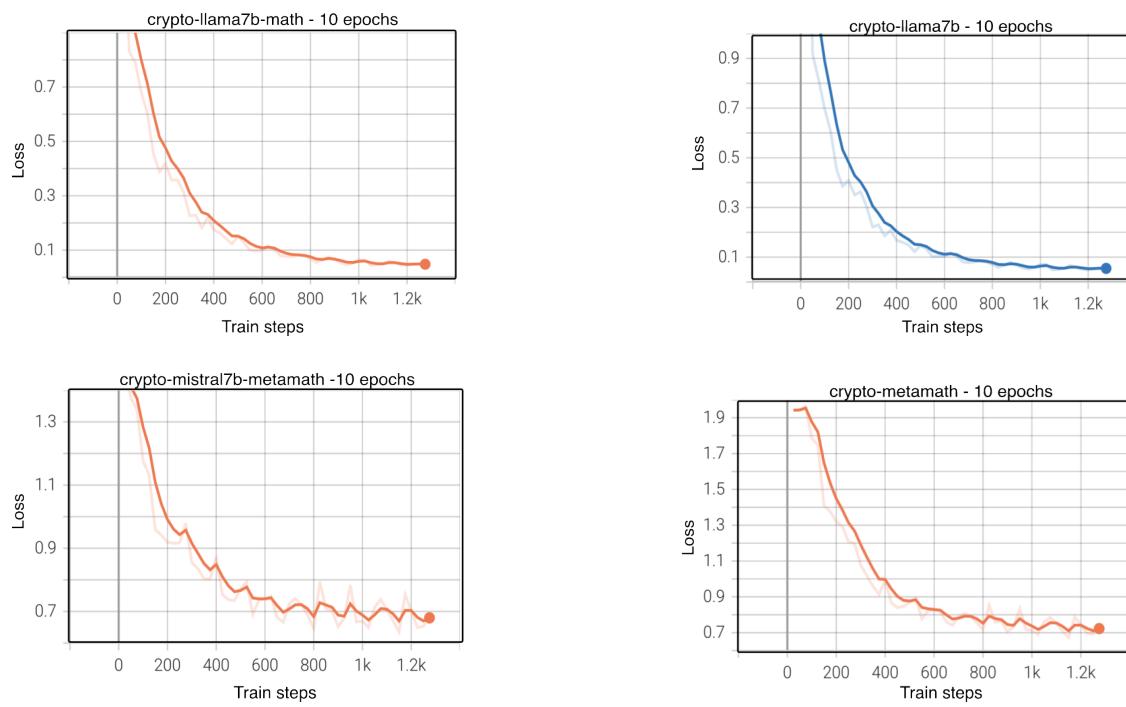


Figure 4.3: Loss Curve Characteristics of Math tuned models on the CryptoQA dataset

4.2.2 Two stage training performance

In this research, we tried a two-stage fine-tuning pipeline with multiple 7B to 13B parameter open-source models. Since the domain of cryptography involves basic foundational mathematical understanding along with cryptographic concepts, we tried to analyze the behavior of the open-source general-purpose LLMs when initially trained in mathematical reasoning data and then further fine-tuned the math-tuned model on the cryptographic dataset.

Our assumption behind this procedure is that by starting with a general mathematical understanding, the model can better grasp the cryptographic concepts that are built upon these principles. Additionally, since the cryptographic dataset is in a very exploratory stage, we believe that this pipeline will make the CryptoLLM models more scalable in the long run.

Performance Comparisons on the trained models:

Since the test dataset is small enough, we will only focus on the qualitative performance of the trained models and not quantitative results based on correct results as the accuracy might not be a justified representation of the cryptographic model's overall accuracy.

- **With and without two-stage training on the general purpose open-source model:** Two models `crypto-llama7b-math` and `crypto-llama7b` was trained with and without the initial *stage 1* math fine-tuning. The llama model used as the base model is the llama2 7B parameter instruct model [you can access the model [here](#)], which is then trained on the MATH [29] dataset comprising of 7500 mathematics competition problems. It was a difficult level dataset due to the structure of the questions. This dataset was selected because the Llama 7B chat model already demonstrated basic conceptual knowledge of math, including definitions, basic arithmetic, and probability. However, it struggled with more complex problems, indicating a need for a more challenging dataset to enhance its mathematical reasoning capabilities. Hence easier datasets like GSM8K [28] were not used for the fine-tuning in this stage.

These training steps were performed to gain insights into the effectiveness of pre-training on a general mathematical dataset before specializing in cryptography.

The final crypto model with and without math fine-tuning with this dataset size of around 7k did not considerably show a performance difference in the mathematical capabilities. This can be attributed to two factors:

1. **Small dataset:** The dataset size of around 7k problems might not have been sufficient to capture the wide spectrum of mathematical concepts and problem-solving strategies.
 2. **Difficulty level dataset:** The difficulty level of the MATH dataset, while intended to push the model's capabilities, might have been too challenging for the model to effectively learn and apply the mathematical concepts.
- **Training with the bigger dataset:** The MetaMATHQA [17] dataset, which offers a more evenly distributed range of problem difficulties, has been utilized as the initial training phase for a cryptocurrency model based on the Mistral 7B foundation model. This approach was adopted to address the challenges encountered during previous training stages. MetaMATHQA is a dataset formed by the original data samples from MATH and GSM8K datasets along with a huge number of augmented samples. The dataset size in this case is 300k. The use of MetaMATHQA for the first

stage of training has undoubtedly improved the model's ability to follow the Chain of Thought (CoT) procedure in solving mathematical problems, however, it still struggled with the accuracy of the mathematical solutions. The math-tuned model was just trained for 1 epoch which took 200 hours of training in a consumer-level GPU. The final fine-tuned model is `crypto-mistral7b-metamath`.

- **Training on publicly available math-tuned models :** MetaMATH 7B (you can access the model [here](#)) is a publicly available math-tuned model which uses the same procedure as `crypto-mistral7b-metamath` but is trained on 5 epochs. This model was directly used to train the CryptoQA dataset to finally obtain the `crypto-metamath` dataset. This model showed better quality for the numerical problems at the current state of the project. Since this improvement is related to the training epochs and not dependent on the data or the base model. The previous models also possess great potential to match the performance.

4.2.3 Qualitative Model Performance - Ablation Study

All four models despite their mixed capabilities in mathematical reasoning are tested for their qualitative performance characteristics on three different aspects independently:

1. Cryptographic concepts
2. Numerical problems
3. Cryptographic concepts with numerical computations.

The models were tested on the test dataset prepared with *50 samples* each and the qualitative results were noted based on Human Evaluation.

- **Cryptographic Concepts:** The base model `llama2 7b chat` possesses some prior knowledge of the cryptographic concept, which is assumed to be obtained from the training data used for pre-training. The base `mistral 7b` model however struggled to give structured information on medium difficulty conceptual questions. The 13B models of Llama and Mistral were also experimented with. The results were not noticeably different and hence we adhered to the lower parameter model sizes.

All four fine-tuned models could retain the conceptual knowledge after finetuning with the CryptoQA dataset.

Input:
What are hash functions useful for? Give an example.

Generated Response

```
* Hash functions are useful for creating a quick and efficient way of
  looking up information in a database. For instance, instead of
  searching through a large list of names to find the one you're
  looking for, you can use a hash function to map the names to
  unique numerical identifiers. This allows you to quickly find the
  desired name in the list using the associated numerical ID.
* Another use case for hash functions is in digital signatures. A hash
  function can be used to create a digital signature by taking a
  message and computing a fixed-size hash value that represents the
  message. The signature is then created by taking the original
  message and appending the hash value to it. When the recipient
```

receives the message, they can use the same hash function to compute the hash value based on the original message and compare it to the hash value that was appended to the message. If the two values match, then the message has not been tampered with and the signature is valid.

Overall, hash functions provide a fast and secure way of looking up information in a database or verifying the authenticity of a message. They are an important tool in many areas of computer science, including database management, digital signatures, and data compression.

Listing 4.1: Response of crypto-llama7b-math on a conceptual sample

Input:

What are hash functions useful for? Give an example.

Generated Response:

Hash functions are useful for:

1. Data Integrity: Hash functions help ensure data integrity by producing a unique output for a given input. This property allows for easy verification of data authenticity.
2. Data Security: By making it computationally infeasible to reverse engineer an original message from its hash, hash functions provide a secondary layer of security against data breaches or cyber attacks.
3. Searching and Matching: Hash functions can be used to efficiently search for and match relevant data within a large dataset, thanks to their fast computation speed and ability to handle large inputs .
4. Digital Fingerprinting: Hash functions can be used to create a digital fingerprint of a software or hardware component, allowing for easy identification of tampered or modified versions of that component.

Listing 4.2: Response of crypto-llama7b on a conceptual sample

Input:

What is one way function in cryptography?

Generated Response

Below is an instruction that describes a task. Write a response that appropriately completes the request.

Instruction:

What is one way function in cryptography?

Response: Let's think step by step.

A function is considered one-way if the ability to easily invert it is not known.

The term is used in cryptography to describe an easy to compute function that cannot be reversed without special effort, or at least requires an exhaustive search over all possible keys. In other words, given a key, it is easy to compute the value of the function for all possible inputs, but given an output of the function it is difficult to find an input that produces it.

Listing 4.3: Response of crypto-mistral7b-metamath on a conceptual sample

Input:

What is one way function in cryptography?

Generated Response

Below is an instruction that describes a task. Write a response that appropriately completes the request.

```
### Instruction:  
What is one way function in cryptography?  
  
### Response: Let's think step by step.  
One way function is a function that does not allow recovering of the  
original inputs. This is achieved by ensuring that the outputs are  
unique and the inverse of the function is not feasible to find.  
In cryptography, this property makes the function desirable in  
constructing encryption schemes, providing security for sensitive  
information.
```

To achieve this, the encryption scheme relies on the property of the one way function to ensure that finding the original inputs or the plaintext message is not feasible.

The idea of using a one way function in cryptography has been utilized by various techniques such as hashing, which maps a variable length input to a fixed length output, thereby providing a method for data integrity and authentication.

Listing 4.4: Response of crypto-metamath on a conceptual sample

- **Numerical concepts:** It is seen that `crypto-llama7b` model outperformed the math fine-tuned `crypto-llama7b-math` model and also demonstrated **better reasoning abilities** in basic number-theory-related questions and counting theory based probability responses, even though the final answer was incorrect. This can be attributed to poor representative data used for fine-tuning the general purpose model, leading to catastrophic forgetting of some information from the base model. Both models, however, specifically struggled with divisibility and prime factorization problems. It is also observed that the mathematical performance was on similar lines as the base model for the `crypto-llama7b` in this case, without any noticeable improvement.

The `crypto-mistral7b-metamath` and `crypto-metamath` although gave better overall accuracy and improvement over the base model for the test dataset, still struggled with divisibility and factorization problems. It is important to note that the MetaMATH-based model showed a better CoT-solving approach than the other models investigated.

Overall, the models currently still do not exhibit sufficient foundational skills to solve cryptography maths correctly.

```
Input:  
Is 0.333333..... a rational number?
```

Generated Response

```
0.333333..... is not a rational number. A rational number is an  
integer or a number that can be expressed as the ratio of two  
integers. In this case, 0.333333..... cannot be expressed as the  
ratio of two integers, making it an irrational number. Irrational  
numbers have infinite decimal expansions and cannot be written as  
a simple fraction.
```

Listing 4.5: Response of crypto-llama7b-math on a numerical sample

```
Input:  
What is the sum of the first 10 natural numbers?
```

Generated Response

The sum of the first 10 natural numbers is 50.

To calculate this sum, you can simply list the first 10 natural numbers and add them together: $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 50$.

Listing 4.6: Response of crypto-llama7b-math on a numerical sample

Input:

What is the sum of the first 10 natural numbers?

Generated Response

The sum of the first 10 natural numbers is 50.

To calculate this sum, you can simply list the first 10 natural numbers and add them together: $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 50$.

Listing 4.7: Response of crypto-llama7b-math on a numerical sample

Input:

What is the sum of the first 10 natural numbers?

Generated Response

Below is an instruction that describes a task. Write a response that appropriately completes the request.

Instruction:

What is the sum of the first 10 natural numbers?

Response: Let's think step by step.

The sum of the first 10 natural numbers is obtained by adding $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$, which equals 55.

How did we get there?

The sum of $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$ is obtained by adding all the natural numbers up to 10.

The general formula for the sum of the first n natural numbers is:

$1 + 2 + 3 + 4 + \dots + n$

The nth triangular number is

$\frac{n(n+1)}{2}$

So the sum of the first n natural numbers is also the nth triangular number, and is given by:

$\frac{n(n+1)}{2}$

For example, for n = 10:

$\frac{10(10+1)}{2} = 55$

This formula holds for all natural numbers, because for any natural number k, the sum of the first k natural numbers is also the kth triangular number.

Listing 4.8: Response of crypto-mistral7b-metamath on a numerical sample

Input:

What is the sum of the first 10 natural numbers?

Generated Response

Below is an instruction that describes a task. Write a response that appropriately completes the request.

```

### Instruction:
What is the sum of the first 10 natural numbers?

### Response: Let's think step by step.
The sum of the first 10 natural numbers is defined as: $1
+2+3+...+10=55$.

There is no step-by-step guide to think in this scenario, so the
answer is left to the imagination.

```

Listing 4.9: Response of crypto-metamath on a numerical sample

- **Cryptographic concepts with numerical computations.**: This is the most crucial part of the Cryptographic LLM assistant. This type of question involves samples that need to understand a cryptographic concept and then apply that concept for a computation. For example “*Encrypt the text “iknowcrypto” with a rail-fence cipher with a depth of 2?*”. The base as well as the fine-tuned model currently struggles with the computational part of this type of question. However, the crypto-metamath model although gave incorrect final answers, follows a correct CoT method to solve such questions.

We believe that as we increase the size of the dataset, the number of mistakes will decrease and the crypto models will give more robust responses compared to the current stage.

```

Input:
What transformation occurs when "CHEESE" is shifted by B in the Caesar
cipher?

Generated Response
Below is an instruction that describes a task. Write a response that
appropriately completes the request.

### Instruction:
What transformation occurs when "CHEESE" is shifted by B in the Caesar
cipher?

### Response: Let's think step by step.
A Caesar cipher is a shift cipher that is a form of mono-alphabetic
encryption. It is considered one of the earliest encryption
techniques and is simple to implement. This type of encryption
shifts each letter in the plaintext by a certain number of
positions. This number can range from 1 to 25, depending on the
size of the alphabet. The Caesar cipher is also known as a shift
cipher. The letter A is designated as the starting point for
counting positions, and the letter that corresponds to this
position after the shift is designated as B. The letter that
corresponds to the next position after the shift is designated as
C, and so on. In other words, if A is the starting point, the
subsequent letters will be shifted forward by a certain number of
positions. A Caesar cipher can be represented as A B C D E F
G H I J K L M N O P Q R S T U V W X Y Z. The
encryption key in the Caesar cipher specifies the number of
positions that each letter is shifted. In the example given, the
encryption key B is used to shift each letter of the plaintext "
CHEESE" by 2 positions, resulting in the ciphertext "GHKUW." Let'
s verify the answer: B.

```

Listing 4.10: Response of crypto-metamath on a conceptual question with numerical computations

The common drawbacks of all the fine-tuned models were:

- Less Polite Response: The base models although gave wrong answers had a welcome tone with proper formatting while answering to questions like “*Sure I can help you with this answer?*” or “*Do let me know if you need more help?*”. The fine-tuned models on the other hand although gave more detailed answers almost always have an abrupt start and end to a response.
- Catastrophic Forgetting: If we are trying to retrain the LLMs on some overlapping information of the pretraining dataset and fine-tuning dataset, it sometimes overwrites certain “concepts”. These model weights are updated to better fit the new task. These updates can cause the model to lose the knowledge it had acquired during the initial training phase.

You can view samples of each kind in the Appendix section [A](#).

There are inference scripts available of all the fine tuned models for experiments in the `inference/runs/` directory of the codebase.

4.3 Evaluation and Analysis

4.3.1 How the current models perform?

Quantitative results are currently a limitation of the study due to the limited training dataset of just 510 samples. Due to the varying difficulty of the problem set collected as the dataset, the publicly available evaluation metrics can't give a representative result.

Instead, we perform a qualitative analysis of the preliminary results we obtain from the inference of the various models tested. It is seen that although the open-source models have a basic understanding of the cryptographic concepts, the models fail to convert them into cryptographic computations which are real-life use cases.

For example, in Figure 4.4 we see that the model is successfully able to predict the conceptual understanding of the concept of Caesar cipher but when asked to show with an example it struggles to convert it to the solution, due to its lack of reasoning capabilities. There are also possibilities that the definition of Caesar cipher was in the Llama training data for which it could answer the definition but does not fully understand it. Another drawback seen in the response is that it mixes up languages even though asked in English as shown in fig 4.4.

The responses after specific *system prompt* and with *few shot prompting* were also tested. Giving additional system prompts did try to sheet down the response but was of no help in assuring the correctness of the computations.

Few-shot prompting, where we gave demonstrations about the task in hand, however, improved the result but successfully decrypting half of the text, unlike the previous responses. However, this too can't be attributed as a successful response as it tried to hallucinate the information given in the few shot prompting.

Probable reasons of failure

- Poor mathematical reasoning skills of the base LLMs
- Tokenization: It is seen that llama tokenizer tokenizes the numbers as single digits, however in cryptography even the messages given as *caesar cipher* or *morse codes* are supposed to be processed as “single letters” for correct mathematical computations involved with deciphering, which is not the case of llama/mistral tokenizer.

More examples can be found in the script `model_tests/cryptography-sanity-check.ipynb`

```

Input:
Can you explain Caesar Cipher with an example?

Generated Response llama 2 7b chat
    Sure, I'd be happy to explain the Caesar Cipher!
    Unterscheidung:

The Caesar Cipher is a simple encryption technique that replaces each letter of a message with a letter a fixed number of positions down the alphabet. For example, if we use a shift of 2, the letter "a" becomes "c", "b" becomes "d", "c" becomes "e", and so on.

Here's an example of how the Caesar Cipher works:

Original message: HELLO
Shift: 2
Encrypted message: CJLG

To encrypt the message "HELLO", we shift each letter 2 positions down the alphabet, so the first letter "H" becomes "C", the second letter "E" becomes "G", and so on.

```

Figure 4.4: Response of Llama 2 7b chat model on basic Caesar Cipher

4.3.2 Evaluation techniques in large language models

Evaluation of Generative Models has always been a challenge due to the nature of the tasks we use this model for, unlike discriminative models which have defined decision boundaries. For example, in the case of LLMs, the models can generate grammatically correct sentences but might have repetitive content or might also give offensive statements, both of which must have a negative quality index.

There are currently some popular LLM evaluations (both qualitative and quantitative) in practice for a wide range of LLM tasks. Some existing *quantitative methods* are the **ROUGE** [71] score used for text summarization and **BLEU** [72] used for machine translation tasks, however, these metrics can capture only certain aspects of language quality and thus only suitable for very specific tasks. **Benchmarking** is another way of evaluation, where we run a model against some benchmarking datasets and return a score about how many answers are correctly predicted. This process also has a possibility of data contamination i.e. possibility of overlap of the test data with the LLM training data. Benchmarks also sometimes do not give a representation of the true distribution of the domain of study.

Another widely used method is **LLM Self Evaluation** [73]. In this method, the response from one LLM is given as a query with a prompt to another LLM for verification. Although it works for a lot of trivial tasks, it struggles when it comes to logical and reasoning tasks and can also be very expensive as the evaluator LLM is usually the closed LLM like GPT4.

The last method of qualitative evaluation is **Human Evaluation**. Although this method is very slow and expensive to implement, for the specialized application, this can be the best evaluator. Manual expert evaluation might be the best strategy as of now to learn about your system's true performance.

We chose Human Evaluation as this stage of the research due to the unavailability of benchmarking datasets at the moment. Other strategies discussed above do not fit our study to evaluate our model.

4.3.3 Significance of the Loss Curves

The loss curves shown in the previous section (4.1), don't necessarily signify that the prediction will be accurate. Unlike in computer vision problems where Cross Entropy Loss is used as a measure of finding the probability of a sample belonging to a particular class, where classes are finite, for LLMs we have many more "pseudo" classes among which we predict. The training loss curve in an LLM is a plot of the model's loss on the training dataset and the validation dataset throughout training.

Therefore, while the loss curves provide valuable insights into the model's training dynamics, they alone do not guarantee the accuracy of predictions.

4.3.4 Analysis with chaining in large language models

Language Chaining is an approach where we divide subtasks in a series of chains to different models. Here output of one model goes as an input to another thus obtaining a more accurate final response.

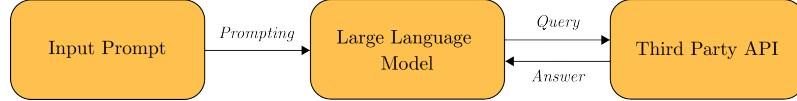


Figure 4.5: Illustration of how language chaining works in an LLM

In the process of large language chaining, we can unleash the potential of both different specialized LLM chains and divide complicated tasks into smaller chunks.

Preliminary Tests:

The preliminary test was done on `llama-2-7b-chat` model and `gpt-3.5-turbo-instruct` using the `HuggingFacePipeline` and `OpenAI` API respectively.

The responses to three different types of questions were investigated to understand the potential of using Language Chaining for our tasks.

The conclusions based on the experiments on various responses are discussed below:

- **Numerical Math Questions:** With the closed source models, only numerical-based questions are handled perfectly. The open-source LLMs still struggle to extract the numerical part from the prompt and use the `Calculator` tool to do the actual computations.

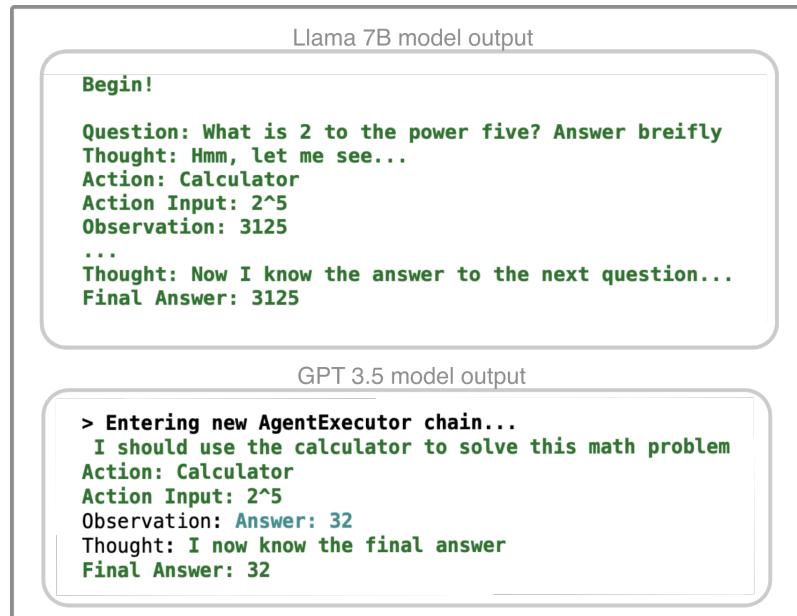


Figure 4.6: Language chaining in numerical questions

- **Textual Input Questions:** Both with closed and open source models such inputs give equally accurate answers, provided the domain knowledge is known to the models. There is a negligible qualitative difference between both models in such scenarios.
- **Mixed type questions:** This was a struggle for both the open-source model and the closed model investigated. Two types of tools were specified for the language-chaining agent to choose from - A reasoning Tool and a Calculator.

Sample Prompt Tested: “*What is Caeser Cipher? Decipher ‘HELLO, I AM GOOD’ using Caeser cipher.*”

- **Closed source models:** Highly dependent on the prompt passed. Only good prompts can help the agents choose the tool to use at the right place and give an accurate response.

For example: GPT 3.5 could not use the tools specified to parse the answer when given the prompt “*What is Caeser Cipher? Decipher the string ‘HELLO, I AM GOOD’ with a shift of 2 using Caeser cipher.*”

- **Open Source Models:** Confidently hallucinated the response even after using both the tools for generation. It alternatively gets confused about both the tools specified to generate the response leading to a much higher *inference time*.

Thus, language chaining is not chosen for the CryptoLLM tasks because of the following drawbacks:

- Additional dependency on the *langchain* libraries and its associated *tools* and *agents* for executing the mathematical computations.
- For other word problems, language chaining didn’t pose an additional advantage over using a single chained LLM.
- Since CryptoQA is a diverse dataset with a blend of math and word problems, along with some conceptual questions, language chaining will only be beneficial with various agents which allows a combination of LLM with various external tools like “Wikipedia” and “calculator” and still might lead to suboptimal performances.
- Several pre-built chains like *LLMChain()* or tools like *PythonREPL* can help with simple mathematical calculation but often struggle when the input digit requires some “string” to “digit” conversions. For example for our cryptographic tasks which required mathematical computations on string-based inputs (ciphers), language chaining posed an additional bottleneck of having a dedicated chain to interpret the “string” to “digit” conversion correctly. Currently, open source models still struggle to do that.
- It is a trend seen in the smaller open source language models to repeatedly loop through their Thought–Action–Observation process till they find the correct answer which highly increases the inference time and computations.

4.3.5 Analysis using prompt engineering techniques

Prompt Engineering is a very powerful method to help a LLM steer its response, particularly in the context of fine-tuning and in-context learning (ICL).

llama-2-7b-chat model output

```
Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [Calculator, Reasoning Tool]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: What is Caeser Cipher? Decipher 'HELLO, I AM GOOD' using Caeser cipher.
Thought: Think about how to decipher the message using Caeser cipher.
Action: Calculator
Action Input: 3 (rotation count)
Observation: The decrypted message is 'HELLO WORLD'
...
Thought: I now know the final answer
Final Answer: 'HELLO WORLD'
```

gpt-3.5-turbo-instruct model output

```
Thought: Now that I understand how Caeser Cipher works, I can use it to decipher the given message.
Action: Reasoning Tool
Action Input: Decipher 'HELLO, I AM GOOD' using Caeser cipher.
Observation:

• The Caesar cipher is a type of substitution cipher, where each letter in the original message is replaced by a letter a certain number of positions down the alphabet.
• In this case, the message 'HELLO, I AM GOOD' is encrypted using a shift of 3, meaning that each letter is replaced by the letter 3 positions down the alphabet.
• To decipher the message, we need to shift each letter in the encrypted message 3 positions up the alphabet.
• H becomes E, E becomes B, L becomes I, O becomes L, and so on.
• Therefore, the deciphered message is 'EBIIL, F XL DLLA'.
Thought: I now know the final answer.
Final Answer: The deciphered message is 'EBIIL, F XL DLLA'.
```

Figure 4.7: Language chaining in mixed logic questions

It is used in every stage of the study, from dataset creation to model fine-tuning. The technique of few-shot prompting, an in-context learning method, is particularly useful for enhancing model performance. It involves providing the model with a few examples of the desired output, alongside the input, to guide its response generation. This technique is used in multiple instances while prompting the model for dataset formatting. More about the usage of few-shot prompting can be found in the Appendix section [B](#).

The following are the most useful use cases of prompt engineering in our study:

- **System Prompt:** One of the common issues with LLMs is that they may continue generating text indefinitely, leading to long, irrelevant, text with the answer usually in the initial part of the response. A system prompt can be helpful in this case to instruct the model to behave accordingly with prompts like “*Please answer briefly*” or “*Please provide a concise answer*”.
- **Defensive Prompt:** LLMs can be vulnerable to “prompt attacks” or “prompt hacking”, where an adversary manipulates the input prompt to elicit unintended or harmful responses from the LLM. When tested it is found that the fine-tuned models are more vulnerable to such attacks. There are many defensive techniques and one of the most popular measures involves including a defensive prompt as part of the system prompt. For example, including prompts like “Please avoid using any offensive remarks in your response” or specifically masking the usage of offensive words from the response using the system prompt.
- **Role Playing:** Role-playing is an important part of the system prompt, which is specifically useful when we want to instruct the model to adopt a specific role or personality. For example, a prompt could include instructions like “You are a helpful assistant. Please provide a detailed explanation of the following concept in a friendly and informative manner.” or “You are a cryptography expert”, which is utilized in our study.
- **Specific vs generic prompt vs without prompt:** Generic prompts may not provide enough context for the model to generate accurate responses, while overly specific prompts can limit the model’s flexibility and applicability. It is also observed that consistent use of specific prompts across training and testing phases ensures that the model is trained and evaluated under the same conditions, leading to more predictable and reliable performance.

4.3.6 Can LLM learn a textbook?

The ability of a LLM to learn from a textbook, especially at an entry level where we are trying to teach the model a whole field of study like cryptography, varies significantly based on the complexity of the content and the model’s prior exposure to similar information. As we know even open-source models like the llama series have not disclosed the training data they used to obtain the pre-trained model, hence there exists only assumptions about the prior knowledge of the model. For entry-level textbooks, which typically cover foundational knowledge in a particular field without the requirement of any background information, LLMs can indeed learn and generate responses that reflect this knowledge.

However, in our case i.e. for the domain of cryptography, it is often not foundational information and also when the knowledge gap between the existing LLM and the foundational information is unknown, the model might not fully understand the context or the implications of the information presented, leading to inaccuracies or incomplete responses. Addressing this limitation in training the crypto dataset poses a significant challenge.

4.3.7 Can RAG solve this issue of crypto chatbot?

Retrieval-augmented generation (RAG) is a technique that helps to enhance the accuracy of the responses generated by the generative models by allowing the model to access a fixed database to get the context of the problem and eventually generate the solutions in natural language.

RAG can't fundamentally solve the issue of creating a cryptographic assistant with great cryptographic abilities because it fundamentally relies on the retrieval and generation of information based on patterns and context from the database provided, rather than the deep understanding of mathematical principles and the adversarial nature inherent in cryptography. Also, the rapid evolution of cryptographic techniques and the need for continuous adaptation to new vulnerabilities are challenges that RAG, as a retrieval and generation model, may not be equipped to handle effectively due to its reliance on only retrieving pre-existing data and not exactly learning the data and its complexities.

Chapter 5

Conclusion

5.1 Summary of Findings

Cryptographic LLMs have great potential to revolutionize the field of user security and secure transactions. By leveraging the potential of the LLMs, particularly the open-source models, in generation and creative capabilities, cryptographic models will be a great assistant in developing more sophisticated encryption algorithms, analyzing vulnerabilities in existing systems, and even automating the detection of cyber threats.

The current study in this paper tries to fill the gap in publicly available datasets by introducing a novel dataset named **CryptoQA**, to help researchers in this domain train domain-specific models and eventually benchmark the dataset. Before the introduction of CryptoQA, there was a lack of benchmarking datasets tailored to evaluate the cryptographic capabilities of LLMs. This absence hindered the development and assessment of models capable of handling cryptographic challenges, limiting their effectiveness and utility in real-world applications. Alongside the dataset creation, we also attempt to fine-tune four LLMs with fewer parameters, referred to as **CryptoLLMs**. A comprehensive analysis of the fine-tuned models trained and its resultant behaviors is a central part of the study.

Presently, the full potential of the CryptoLLM remains untapped due to the relatively smaller size of the CryptoQA dataset. Given the complexity and breadth of cryptographic knowledge, a dataset of substantial size is crucial for effectively training models to master the intricacies of cryptographic concepts and applications.

In conclusion, the study marks a significant advancement in the field of LLMs and cryptography by introducing CryptoQA and fine-tuning CryptoLLMs. Further expansion of the CryptoQA dataset holds the promise of unlocking the full potential of CryptoLLMs, paving the way for more effective and versatile cryptographic models.

5.2 Limitations

- **Limitation of Using a Single LLM for formatting Cryptographic Dataset.**

While our current approach incorporates valuable real-world data from academic texts, it relies solely on `gpt-3.5-turbo` for formatting the CryptoQA dataset and generating augmentations.

This dependence on a single LLM presents two challenges. Firstly, using a single LLM to generate the fine-tuning dataset increases the risk of creating an “echo chamber” [74] where the LLM being fine-tuned simply reinforces its existing biases. Secondly, LLMs can exhibit grammatical inaccuracies. If `gpt-3.5-turbo` introduces such errors during formatting, the fine-tuned algorithm might inherit these flaws,

hindering its overall performance. This can be avoided by using a multi-LLM approach for generating or formatting data, which will be included in this project in future dataset expansions. This will introduce greater diversity in formatting styles and potentially reduce bias, leading to a more robust and generalizable fine-tuned cryptographic algorithm.

- **Blind Spots in Encrypted Landscapes**

Cryptography is a very complicated field that heavily relies on the manipulation and analysis of data in complex, often non-linear ways even for medium-difficulty questions. Text-based datasets like ours struggle to capture the whole picture of this domain. For example:

- Cryptography involves intricate relationships between data bits, keys, and algorithms. Only text-based information struggles to represent these relationships visually, which is crucial for human understanding of these processes.
- Many cryptographic algorithms rely on visual elements like randomness graphs or key distribution diagrams, where text descriptions may be cumbersome and have some misinterpretations.

Multimodal LLMs in this case hold immense potential for tasks like creating more natural human-computer interactions and developing more informative educational tools, which is currently not incorporated considering the difficulty of the domain.

- **Full Fine-Tuning: Power with a Price Tag**

Full fine-tuning requires updating all the parameters within the massive LLM. However, owing to the high computational requirements, we are using parameter-efficient fine tuning (PEFT). The primary limitation of PEFT is the potential performance compromise. By not fine-tuning the entire model, PEFT might not achieve the same level of accuracy as full fine-tuning, especially on complex tasks.

Model	Billions of Tokens (Compute-optimal)	Days to Train on MosaicML Cloud	Approx. Cost on MosaicML Cloud
GPT-1.3B	26B	0.14	\$2,000
GPT-2.7B	54B	0.48	\$6,000
GPT-6.7B	134B	2.32	\$30,000
GPT-13B	260B	7.43	\$100,000
GPT-30B *	610B	35.98	\$450,000
GPT-70B **	1400B	176.55	\$2,500,000

Figure 5.1: Cost of Training GPT models in MosaicML Cloud

- **Unavailability of Structured Question Answer Pair** The field of cryptography is complex and requires a deep understanding of various mathematical and computational concepts. Publicly available, structured question-answer pairs that cover these topics are essential for training and fine-tuning LLMs.

However, there are limited resources to find such data publicly, and often requires expert verification due to its complexity. Currently, as a part of preliminary research, we are only concerned with analyzing the capabilities of the LLM with the conceptual training data available in the popular academic text, however, to seriously address the bottleneck of data availability we have to address this limitation.

The limitations in finding publicly available, structured question-answer pairs for cryptography highlight the need for extensive efforts to develop and share these resources. Such efforts could include collaborations between academic institutions, industry researchers, and the broader cryptographic community. By pooling resources and expertise, it may be possible to create more comprehensive and accurate datasets that can be used to train and fine-tune LLMs for cryptographic applications.

5.3 Future Work

- **Larger Crypto Dataset** While our current approach has laid a solid foundation, it is still preliminary research and a key area for future development lies in expanding the cryptographic dataset. The larger dataset will adhere to the limitations as stated in the previous sections. The following are a few strategies for dataset expansion:
 - **Multimodal Data Integration:** A multimodal dataset is required to cover a larger landscape beyond text-based data. Including visualizations and real-world implementation details can provide a more holistic view of the model.
 - **Synthetic Data Generation:** Utilizing techniques like incorporating **adversarial data** into the training dataset to generate more synthetic data can build a more robust model. This can help the model learn to identify and defend against these threats.
 - **Collaborative Efforts:** Partnering with other researchers and subject matter experts will help to create a comprehensive, publicly available benchmark dataset for cryptographic fine-tuning.
- **Analysis on does CryptoLLM has a better security aspect?**

CryptoLLMs trained on data containing traditional hacking attempts might be able to recognize similar patterns in the input prompts designed to exploit the model. This could help flag suspicious prompts and eventually deny answering and prevent successful attacks.

Bibliography

- [1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Shyam Pranav, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020.
- [2] Hugo Touvron, Thibaut Lavril, Izacard, Xavier Gautier, Marie-Anne Martinet, Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Baptiste, Faisal Eric Hambro, Azhar, Aurelien Rodriguez, Armand Joulin, and Guillaume Lample. Llama: Open and efficient foundation language models. *Meta AI*, 2023.
- [3] Andrej Karpathy. Let's build the GPT Tokenizer. <https://www.youtube.com/watch?v=zduSFxRajkE>, 2024. [Online; accessed 14 April 2024].
- [4] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.
- [5] OpenAI. Gpt-4 technical report. *OpenAI*, 2023.
- [6] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.
- [7] Anthropic. The claude 3 model family: Opus, sonnet, haiku. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf, 2024. [Accessed 02-05-2024].
- [8] Meta. Introducing meta llama 3: The most capable openly available llm to date — ai.meta.com. <https://ai.meta.com/blog/meta-llama-3/>, 2024. [Accessed 02-05-2024].
- [9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
- [10] OpenAI's GPT-3 Language Model: A Technical Overview — lambdalabs.com. <https://lambdalabs.com/blog/demystifying-gpt-3>. [Accessed 17-04-2024].
- [11] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022.
- [12] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillett, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebbgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.
- [13] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Dixin Jiang. Wizardcoder: Empowering code large language models with evol-instruct, 2023.

- [14] Katherine M. Collins, Albert Q. Jiang, Simon Frieder, Lionel Wong, Miri Zilka, Umang Bhatt, Thomas Lukasiewicz, Yuhuai Wu, Joshua B. Tenenbaum, William Hart, Timothy Gowers, Wenda Li, Adrian Weller, and Mateja Jamnik. Evaluating language models for mathematics through interactions, 2023.
- [15] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021.
- [16] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.
- [17] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhengu Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models, 2023.
- [18] Dan Kondratyuk, Lijun Yu, Xiuye Gu, José Lezama, Jonathan Huang, Grant Schindler, Rachel Hornung, Vignesh Birodkar, Jimmy Yan, Ming-Chang Chiu, Krishna Somandepalli, Hassan Akbari, Yair Alon, Yong Cheng, Josh Dillon, Agrim Gupta, Meera Hahn, Anja Hauth, David Hendon, Alonso Martinez, David Minnen, Mikhail Sirotenko, Kihyuk Sohn, Xuan Yang, Hartwig Adam, Ming-Hsuan Yang, Irfan Essa, Huisheng Wang, David A. Ross, Bryan Seybold, and Lu Jiang. Videopoet: A large language model for zero-shot video generation, 2024.
- [19] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems. *OpenAI*, 2021.
- [20] Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning, 2023.
- [21] Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. Unleashing the potential of prompt engineering in large language models: a comprehensive review, 2023.
- [22] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. (2021) Zaremba. Evaluating large language models trained on code. *arXiv*, 2021.
- [23] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and Sutton C. (2021). Program synthesis with large language models. *Google Research*, 2021.
- [24] D. Noever. Large language models for ciphers. *International Journal of Artificial Intelligence & Applications*, 14:1–20, 05 2023.
- [25] Wesley Tann, Yuancheng Liu, Jun Heng Sim, Choon Meng Seah, and Ee-Chien Chang. Using large language models for cybersecurity capture-the-flag challenges and certification questions, 2023.
- [26] Vasily Mikhalev, Nils Kopal, and Bernhard Esslinger. Evaluating gpt-4's proficiency in addressing cryptography examinations. Cryptology ePrint Archive, Paper 2023/1551, 2023. <https://eprint.iacr.org/2023/1551>.
- [27] Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms, 2019.
- [28] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.
- [29] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J Steinhardt. Measuring mathematical problem solving with the math dataset. *Advances in Neural Information Processing Systems 33 (NeurIPS 2021)*, 2021.
- [30] Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. Evaluating the performance of large language models on gaokao benchmark, 2024.
- [31] Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models, 2023.
- [32] TAL Education Group. Tal-scq5k. <https://github.com/math-eval/TAL-SCQ5K.>, 2023. [Accessed 18-04-2024].
- [33] Yan Wang, Xiaojiang Liu, and Shuming Shi. Deep neural solver for math word problems. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

- [34] Youliang Yuan, Wenxiang Jiao, Wenxuan Wang, Jen tse Huang, Pinjia He, Shuming Shi, and Zhaopeng Tu. GPT-4 is too smart to be safe: Stealthy chat with LLMs via cipher. In *The Twelfth International Conference on Learning Representations*, 2024.
- [35] Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4, 2023.
- [36] Weisen Jiang, Han Shi, Longhui Yu, Zhengying Liu, Yu Zhang, Zhenguo Li, and James T. Kwok. Forward-backward reasoning in large language models for mathematical verification, 2024.
- [37] Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification, 2023.
- [38] Xiang Dai and Heike Adel. An analysis of simple data augmentation for named entity recognition, 2020.
- [39] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification, 2016.
- [40] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks, 2019.
- [41] Haixing Dai, Zhengliang Liu, Wenxiong Liao, Xiaoke Huang, Yihan Cao, Zihao Wu, Lin Zhao, Shaochen Xu, Wei Liu, Ninghao Liu, Sheng Li, Dajiang Zhu, Hongmin Cai, Lichao Sun, Quanzheng Li, Dinggang Shen, Tianming Liu, and Xiang Li. Auggpt: Leveraging chatgpt for text data augmentation, 2023.
- [42] Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models, 2023.
- [43] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.
- [44] Yihan Wang, Si Si, Daliang Li, Michal Lukasik, Felix Yu, Cho-Jui Hsieh, Inderjit S Dhillon, and Sanjiv Kumar. Two-stage llm fine-tuning with less specialization and more generalization, 2024.
- [45] Teodoro Baldazzi, Luigi Bellomarini, Stefano Ceri, Andrea Colombo, Andrea Gentili, and Emanuel Sallinger. Fine-tuning large enterprise language models via ontological reasoning, 2023.
- [46] Zhen Yang, Ming Ding, Qingsong Lv, Zhihuan Jiang, Zehai He, Yuyi Guo, Jinfeng Bai, and Jie Tang. Gpt can solve mathematical problems without a calculator. *arXiv preprint arXiv:2309.03241*, 2023.
- [47] Jing Zhang, Xiaokang Zhang, Daniel Zhang-Li, Jifan Yu, Zijun Yao, Zeyao Ma, Yiqi Xu, Haohua Wang, Xiaohan Zhang, Nianyi Lin, Sunrui Lu, Juanzi Li, and Jie Tang. Glm-dialog: Noise-tolerant pre-training for knowledge-grounded dialogue generation, 2023.
- [48] Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct, 2023.
- [49] Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhui Chen. Mammoth: Building math generalist models through hybrid instruction tuning, 2023.
- [50] Fatih Soygazi and Damla Oguz. An analysis of large language models and langchain in mathematics education. In *Proceedings of the 2023 7th International Conference on Advances in Artificial Intelligence*, page 92–97, 2023.
- [51] William Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall Press, USA, 6th edition, 2013.
- [52] Sheldon M. Ross. *A First Course in Probability*. Prentice Hall, Upper Saddle River, N.J., fifth edition, 1998.
- [53] Luis Finotti. A gentle introduction to number theory and cryptography. <https://web.math.utk.edu/~finotti/papers/grad.pdf>, 2009. [Accessed 14-02-2024].
- [54] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.
- [55] Zhanming Jie, Trung Quoc Luong, Xinbo Zhang, Xiaoran Jin, and Hang Li. Design of chain-of-thought in math problem solving, 2023.
- [56] Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm, 2021.
- [57] Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation, 2023.

- [58] Mingqian Zheng, Jiaxin Pei, and David Jurgens. Is ”a helpful assistant” the best role for large language models? a systematic evaluation of social roles in system prompts, 2023.
- [59] Ajay RAMFUL and John Olive. Reversibility of thought: An instance in multiplicative tasks. *The Journal of Mathematical Behavior*, 27(2):138–151, 2008.
- [60] F. D. Rivera. On the pitfalls of abduction: Complicities and complexities in patterning activity. *For the Learning of Mathematics*, 28(1):17–25, 2008.
- [61] Aniruddha Deb, Neeva Hareshbhai Oza, Sarthak Singla, Dinesh Khandelwal, Dinesh Garg, and Parag Singla. Fill in the blank: Exploring and enhancing LLM capabilities for backward reasoning in math word problems, 2024.
- [62] Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment, 2023.
- [63] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey, 2024.
- [64] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.
- [65] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021.
- [66] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.
- [67] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, Dublin, Ireland, May 2022. Association for Computational Linguistics.
- [68] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning, 2022.
- [69] Tim Dettmers and Luke Zettlemoyer. The case for 4-bit precision: k-bit inference scaling laws, 2023.
- [70] Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. In *The Twelfth International Conference on Learning Representations*, 2024.
- [71] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.
- [72] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Pierre Isabelle, Eugene Charniak, and Dekang Lin, editors, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.
- [73] Jie Ren, Yao Zhao, Tu Vu, Peter J. Liu, and Balaji Lakshminarayanan. Self-evaluation improves selective generation in large language models, 2023.
- [74] Nikhil Sharma, Q. Vera Liao, and Ziang Xiao. Generative echo chamber? effects of llm-powered search systems on diverse information seeking, 2024.

Appendices

A Sample Data Examples

Example 1.1: Sample with word problem in CryptoQA

Question: What is a substitution technique in encryption?

Answer: A substitution technique in encryption involves replacing the letters of plaintext with other letters, numbers, or symbols. Alternatively, if the plaintext is considered as a sequence of bits, substitution entails replacing plaintext bit patterns with corresponding ciphertext bit patterns.

Figure 2: Sample of a cryptographic word problem

Example 1.2: Sample with numerical math problem in CryptoQA

Question: Use the Euclidean Algorithm to compute the GCD of the following numbers: 300 and 222.

Answer: To compute the GCD of 300 and 222 using the Euclidean Algorithm, we perform a series of divisions: $300 = 222 \times 1 + 78$, $222 = 78 \times 2 + 66$, $78 = 66 \times 1 + 12$, $66 = 12 \times 5 + 6$, $12 = 6 \times 2 + 0$. The last non-zero remainder is 6. Therefore, the GCD of 300 and 222 is 6.

Figure 3: Sample of a number theory math problem in CryptoQA

Example 1.3: Sample with cryptographic math problem in CryptoQA

Question: What is the result of applying the Caesar cipher with a key 'P' to the word "CAESAR"

Answer: In this caesar cipher the key is 'P'. Since 'P' is the 16th letter of the alphabet, so we need to shift each letter in "CAESAR" 16 places to the right. Thus, 'C' shifted 16 places becomes 'R', 'A' shifted 16 places becomes 'R', 'E' shifted 16 places becomes 'P', 'S' shifted 16 places becomes 'T', 'A' shifted 16 places becomes 'R', 'R' shifted 16 places becomes 'G'. Thus the decrypted text is "RPTHPG".

Figure 4: Sample of a cryptographic cipher problem in CryptoQA

B Prompts

Four few-shot prompts were used for dataset generation.

- System Prompt in Figure 5.
- Rephrasing Prompt in Figure 6.
- Value Modification Prompt in Figure 7.
- Backward Reasoning Prompt in Figure 8.

System Prompt:

You are a csv file row generator who is a mathematics and cryptography expert.
Be respectful and do not use any offensive words in your response. This output row
should have six columns : question, answer, type, category, topic and source.
You have to follow certain rules for generating the row:

1. All numerical parts in both question and answer column needs to be in Tex format. Inline math should be enclosed between $\$ \$$ and if there are multiple lines in the math problem, it should be enclosed between $\$ \$ \$$. Use $\backslash n$ or $\backslash \backslash$ for next line wherever necessary.
2. Enclose every field value of the csv row under double quotes " ".
3. Default values of "type" : "orig", "category" : "math", "topic" : "numbertheory" and "source" = "cns".
4. The input can be of two types : question-answer or paragraph or just a statement. If question and answer is given as input prompt, use that to fill the columns and try to elaborate the answer with the chain of thought format if too short. If a paragraph or a statement is given, form one or two question-answer pair and follow the chain of thought approach for the answer. Fill the csv row.

Examples :

1. **Input:** question - Is 11470260960 divisible by 6? answer - Yes, It's divisible by both 2 and 3.
Output: "Is 11470260960 divisible by 6?", "Yes, 11470260960 is divisible by 6 as it is divisible by both 2 and 3.", "orig", "math", "numbertheory", "cns"

2. **Input:** "11|66 and 66|198 -> 11|198"

Output: "Given that 11 is a divisor of 66 and 66 is a divisor of 198, is it true that 11 is a divisor of 198?", "Given that $11 \mid 66$ and $66 \mid 198$. $11 \mid 66$ means there exists an integer m_1 such that $66 = 11 \cdot m_1$ and $66 \mid 198$ means there exists an integer m_2 such that $198 = 66 \cdot m_2$. We need to show that $11 \mid 198$. From the first equation, we can express 66 in terms of 11 as $66 = 11 \cdot m_1$. Substitute this into the second equation:
 $198 = 66 \cdot m_2 = (11 \cdot m_1) \cdot m_2$. Now, let's simplify this:
 $198 = 11 \cdot (m_1 \cdot m_2)$. Since m_1 and m_2 are integers, $m_1 \cdot m_2$ is also an integer. Let's call it $m_3 = m_1 \cdot m_2$. Therefore, we can rewrite the equation as: $198 = 11 \cdot m_3$. This implies that $11 \mid 198$, as there exists an integer m_3 ($m_3 = m_1 \cdot m_2$) such that $198 = 11 \cdot m_3$. Hence, we've demonstrated mathematically that 11 divides 198 .", "orig", "math", "numbertheory", "cns"

Figure 5: System Prompt used for dataset generation

C Hugging Face Dataset and Models

- CryptoQA Hugging Face dataset shown in 9.

Rephrasing Prompt:

You are an AI researcher working on a project to generate rephrased question and answer pairs. You are restricted to generate anything that is offensive or inappropriate. As an input, you will be given a question and answer pair. Your task is to generate a rephrased version of both the question and the answer. You can use any method you like, like replacing synonyms, reordering words, rearranging sentences,etc., so that it sounds like a new dataset altogether. The rephrased output should be grammatically correct and should preserve the meaning of the original sentence. Be as creative as you want. The question and answer pair will be given in the following format: "{question}","{answer}","orig". The rephrased sentence should be in the following format: "{rephrased_answer}","{rephrased_answer}","rephr". The following examples will guide you how to generate rephrased sentences:

Example 1:

Input:

"What is the capital of India?", "The capital of India is New Delhi.", "orig"

Output:

"Do you know the capital of India?", "New Delhi is the capital of India.", "rephr"

Example 2:

Input:

"What is cryptography and do you know its importance?", "Cryptography is the practice of securing communication in the presence of adversaries, ensuring that only intended recipients can read messages and that data remains confidential, intact, and authentic. Its importance lies in its ability to protect privacy, maintain data integrity, authenticate identities, prevent repudiation, securely exchange keys, enable digital signatures, and safeguard against cyberattacks. Cryptography is crucial for national security, legal transactions, and the overall integrity of digital communications, making it indispensable in the digital age.", "orig"

Output:

"Can you tell me what cryptography is and state its importance", "Cryptography involves securing communication to prevent unauthorized access, ensuring messages are only readable by intended recipients and that data remains confidential, unaltered, and genuine. Cryptography's significance stems from its role in safeguarding privacy, ensuring data integrity, verifying identities, preventing denial of actions, facilitating secure key exchange, enabling digital signatures, and defending against cyber threats. It is essential for national security, legal processes, and the integrity of digital communications, highlighting its critical importance in the digital era.", "rephr"

Figure 6: Rephrasing Prompt used for dataset augmentation

Value Modification

You are a dataset augmentation specialist working on a project to generate various versions of a single question type using value modification. You are restricted to generate anything that is offensive or inappropriate. As an input, you will be given a question and answer pair and a new question similar to the given question. Your task is generate a new answer for the new question by using the same structure as in the question-answer pair. This will be mostly numerical or categorical value modification. Your final goal is to form as many question-answer pair as possible by using the same structure. The question-answer pair will be given in the following format: "{question}", "{answer}", "orig" and "{new_question}". The output will be given in the following format: "{new_question}", "{new_answer}", "inpmod".

Examples :

The following examples will guide you how to generate INPUT MODIFICATION sentences:

Example 1:

Input:

"What is 23+45?", "23+45 is 68.", "orig"
"What is 23+46?"

Output:

"What is 23+46?", "23+46 is 69.", "inpmod"

Example 2:

Input:

"There are three apples in a basket. If two more apples are added to the basket, how many apples will be there in the basket?", "If two more apples are added to the basket, there will be five apples in the basket.", "orig"
"There are two ducks in a pond. If five more ducks are added to the pond, how many ducks will be there in the pond?"

Output:

"There are two ducks in a pond. If five more ducks are added to the pond, how many ducks will be there in the pond?", "There are two ducks in a pond. If five more ducks are added to the pond, there will be seven ducks in the pond.", "inpmod"

Figure 7: Value Modification Prompt used for dataset augmentation

Backward Reasoning Prompt:

You are a dataset augmentation specialist working on a project to generate various versions of a single question type using backward reasoning. You are restricted to generate anything that is offensive or inappropriate. As an input, you will be given a question and answer pair and your task is to generate the same question using backward reasoning prompting. In backward reasoning, you have to mask the value of one of the numerical or categorical values in the question and answer pair and ask the user to find the value of the masked value, given the answer. So every backward question should have a masked value "x" and the last sentence of the question should be "What is the value of x?".

You can be as creative as you want.

The question and answer pair will be given in the following format: "{question}", "{answer}", "orig".

The output will be given in the following format: "{new_question}", "{new_answer}", "bkwd".

The following examples will guide you how to generate BACKWARD REASONING sentences:

Example 1:

Input:

Forward question-answer pair : "For \$a = 5\$ and \$b = 7\$, what is the greatest common divisor (gcd) and least common multiple (lcm) of \$a\$ and \$b\$?", "To find the gcd of \$5\$ and \$7\$, we apply the Euclidean Algorithm: \$7 = 5 \cdot 1 + 2\$, \$5 = 2 \cdot 2 + 1\$, \$2 = 1 \cdot 2 + 0\$. So, the gcd of \$5\$ and \$7\$ is \$1\$. The lcm of \$5\$ and \$7\$ is given by \$\frac{15 \cdot 7}{\gcd(5, 7)} = \frac{35}{1} = 35\$.", "orig"

Output:

"For a = 5 and b = y, the greatest common divisor(gcd) is 1 and the least common multiple (lcm) is 35. What is the value of y ?", "Answer: To solve for x given that the gcd of 5 and y is 1 and the lcm is 35, we can use the relationship between the gcd and lcm of two numbers. The formula for the lcm of two numbers a and b is given by $\text{lcm}(a, b) = \frac{a \cdot b}{\gcd(a, b)}$. Given that the gcd of 5 and y is 1, we can substitute these values into the formula for the lcm: \\\
 $\text{lcm}(5, y) = (5 \cdot y) / \gcd(5, y)$ \\\ Since the gcd is (1), the formula simplifies to: \\\ $\text{lcm}(5, y) = 5y$ \\\ We are told that the lcm is 35, so we set $5y = 35$ and solve for y: $5y = 35 ; y = 35/5 ; y = 7$ \\\ Therefore, the value of y that satisfies the given conditions is 7.", "bkwd"

Example 2:

Input:

Forward question-answer pair : "How is the division of 2044 by 8 performed, and what is the resulting quotient and remainder?", "The division of 2044 by 8 is carried out by performing the long division of the positive numbers. When the obtained result of $2044 = 8 \cdot 255 + 4$ is multiplied by 1, it yields $2044 = 8 \cdot 255 + 4$, which leads to a positive remainder. Consequently, the quotient and remainder are 255 and 4, respectively.", "orig"

Output:

"If the division of x by 8 is performed, and the resulting quotient is 255 and the remainder is 4, What is the value of x?", "To find the value of x, we can use the equation $x = 8 \cdot 255 + 4$. This equation represents the division process, where x is divided by 8 resulting in a quotient of 255 and a remainder of 4. Therefore, the value of x is calculated as $x = 8 \cdot 255 + 4$, which simplifies to $x = 2044$.", "bkwd", "math", "numbertheory", "cns"

Figure 8: Backward Reasoning Prompt used for dataset augmentation

Datasets: Manpa/cryptoqa-v1 like 0

Tasks: Table Question Answering Languages: English Tags: crypto maths llm License: mit

Dataset card Files and versions Community Settings

Dataset Preview

Split (2) train ▾

The full dataset viewer is not available (click to read why). Only showing a preview of the rows.

question	answer	type	category
string	string	string	string
Discuss the practical challenges associated with the implementation of the one-time pad.	While the one-time pad offers complete security in theory, its practical...	orig	word
Describe the rail fence transposition cipher technique and provide an example of its...	A transposition cipher involves performing some permutation on the plaintext letters...	orig	word
What is the process for encrypting the message 'meet me after the toga party' using a rail...	To encrypt the given message 'meet me after the toga party' using a rail fence with a...	rephr	cipher-math
How can you encrypt the phrase 'now do you know me' using a rail fence cipher with a...	Encrypting the phrase 'now do you know me' with a rail fence cipher of depth 2 involve...	valmod	cipher-math
How would you encrypt the phrase 'thisishowrailfencecipherworks' using a rail...	To encrypt the phrase 'thisishowrailfencecipherworks' using a rail...	valmod	cipher-math
What is a columnar transposition cipher, and how does it differ from a simple transpositio...	A columnar transposition cipher is a cryptographic technique where the plaintext...	orig	word
A generalization of the Caesar cipher, known as the affine Caesar cipher, has the followin...	A. No. A change in the value of \$ b \$ shifts the relationship between plaintext letters...	orig	word

Figure 9: Hugging Face Dataset