A Dataset of Ancient Chinese Math Word Problems and an Application for Research in Historic Mathematics

Florian Kessler
Friedrich-Alexander-Universität Erlangen-Nürnberg
florian.kessler@fau.de

Abstract

Solving math word problems, i.e. mathematical problems stated in natural language, has received much attention in the Artificial Intelligence (AI) community over the last years. Unsurprisingly, research has focused on problems stated in contemporary languages. In contrast to this, in this article, we introduce a dataset of math word problems that is extracted from ancient Chinese mathematical texts. The dataset is made available.[1] We report a baseline performance for GPT-4o solving the problems in the dataset using a Program-of-Thought paradigm that translates the mathematical procedures in the original texts into Python code, giving acceptable performance but showing that the model often struggles with understanding the pre-modern language. Finally, we describe how the generated code can be used for research into the history of mathematics, by offering a way to search the texts by abstract operations instead of specific lexemes.

1 Introduction

In recent years, using techniques such as Chain-of-Thought (CoT, Wei et al., 2022) or Program-of-Thought (PoT, Chen et al., 2023) prompting, Large-Language-Models (LLMs) have achieved excellent performance in solving mathematical problems formulated in natural language, spiking renewed interest in this area of artificial intelligence. However, while datasets of such math word problems are available in multiple languages, to our knowledge, all of them are contemporary. How do LLMs cope with ancient math problems, which might both involve terms that are unfamiliar to the model, as well as rely on knowledge about a world that differs significantly from what the model is familiar with? In order to answer this question, we will describe in Section 3 the creation of a database of mathematical problems extracted from ancient Chinese texts, using a semi-automatic approach that utilizes the highly structured nature of the textual material. Subsequently, in Section 4 the performance of GPT-4o in solving the problems in the database will be tested, showing that it is able to derive solutions for around two thirds of the problems in the database, but often struggles with unfamiliar expressions in the technical language of pre-modern Chinese mathematics. Figure 1 shows an overview over the setup discussed in this article.

While solving mathematical problems in modern settings with LLMs is useful for real world applications, solving problems in the style of ancient Chinese mathematical texts in itself is presumably of no interest to any user. Also, in the texts considered here, the problems posed are always accompanied by numerical solutions. Hence, being able to automatically compute a solution does not provide the researcher with any new information, aside from being a convenient way of checking for textual errors. However, being able to test the model's understanding of the problems lays the groundwork for confidently applying LLMs for different downstream research tasks. Since historic Chinese mathematics remains an understudied subject, such assistance is especially valuable. In particular, in this article, we suggest a way to use the output of the model for a type of semantic search. All of the state-of-the-art prompting techniques for solving mathematical problems using LLMs cause the model to output intermediate results, either in the form of natural language reasoning steps (Wei et al., 2022), program code (Chen et al., 2023; Gao et al., 2023) or systems of symbolic equations (He-Yueya et al., 2023). In Section 5, we will show how such output, in our case in the form of Python code, that can be aligned to the original algorithms provided in the texts using our prompting technique, can be used by histori-

[1] https://github.com/notiho/ancient-chinese-math-problems.

Proceedings of the Second Ancient Language Processing Workshop associated with NAACL 2025, pages 40–51
May 4, 2025.

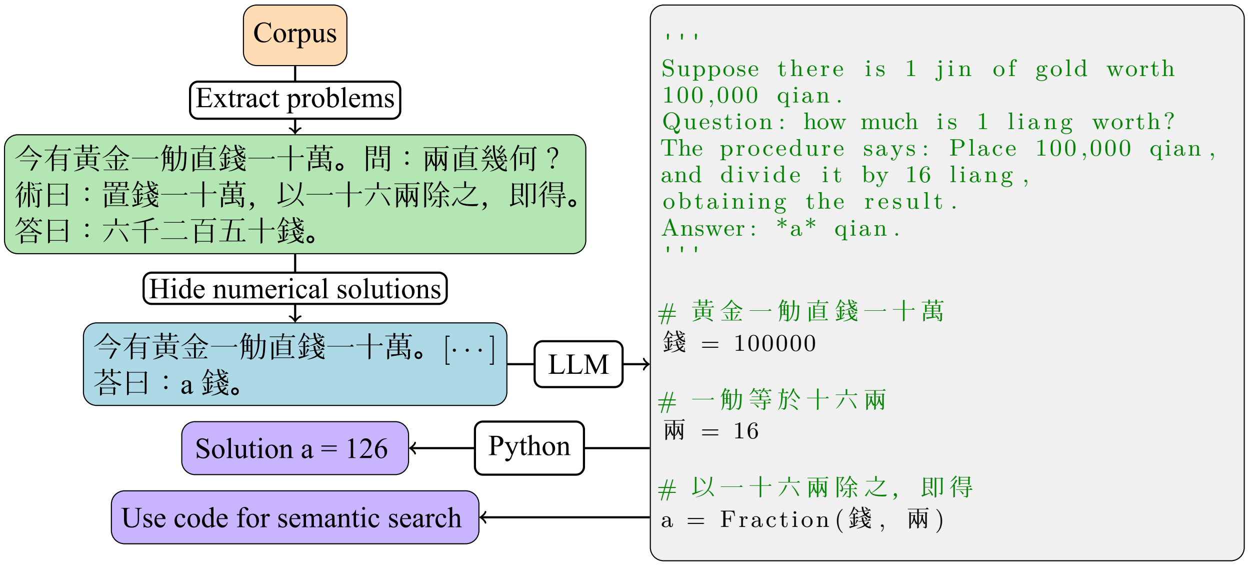

Figure 1: Workflow described in this paper illustrated with problem 18 from chapter 3 of the *Master Sun*

ans of mathematics, by providing a way to search for mathematical contents that abstracts from the language of the texts.

2 Related Work

Solving math word problems using artificial intelligence has been an active area of research for a long time, and accordingly, many datasets of problems with different levels of mathematical difficulty and language have been released (see Ahn et al., 2024 for an overview). To our knowledge, none of these contain problems extracted from premodern works.

In solving math word problems with LLMs, program-of-thought (PoT) prompting, that is, having the model emit program code that computes solutions has been one of the most successful paradigms (Chen et al., 2023; Gao et al., 2023). Recent research has shown that for advanced problems, having the model output symbolic equations instead is beneficial (He-Yueya et al., 2023). However, the problems considered here are relatively simple from the point of view of modern mathematics, and can mostly by easily solved using straightforward arithmetic, although the dataset also includes e.g. procedures for square and cube root extractions and solving what is equivalent to a system of linear equations (Martzloff, 2006: 127-41). Furthermore, the potential of transforming ancient Chinese mathematical procedures into imperative languages has long been recognized by the eminent mathematician and historian of mathematics, Wu Wen-Tsun (e.g. Wu Wen-Tsun, 2019: 121).

3 Building a Dataset

As the basis of the dataset, a collection commonly known as the *Computational Canon in Ten Books* (*Suanjing shishu* 算經十書) was chosen, containing the most important Chinese mathematical texts up to the Tang dynasty (608-907).[2] Having been compiled from mostly much older sources to serve educational purposes in 656 (Keller and Volkov, 2014: 59-63), among its nine surviving works, there are seven that employ a rigid question-answer-procedure pattern to structure their content that is as typical of ancient Chinese mathematics as it is convenient for automatic extraction. The titles of the seven works as well as dating information are shown in Table 1. In the following, the works will be referred to by the shortened translated titles underlined in the table.

In terms of their structure, these works consist of series of usually thematically grouped triples of questions, numerical answers and procedures ("*shu* 術") to compute the answers (see Table 2 for an example).[3]

[2] In fact, one of the titles included is a later apocryphal text that was not included in the original collection (Wu Wenjun and Shen Kangshen, 2000: 82). However, due to its comparable structure and contents, it was nevertheless included in the dataset.

[3] The strict triplet structure is not entirely followed in one of the works, the *Nine Categories*, where it is common to have several pairs of answers and questions followed by a single procedure that solves them all, and there are a few general procedures that precede a series of triplets, the procedures for which are special cases of the general procedure. In the dataset, these general procedures were not included, unless they were referenced by a stub-procedure specific to

Title	Date	Chapters	Problems	Punctuated
"Computational Procedures of Nine Categories" (*Jiuzhang suanshu* 九章算術)	early 1st century	9	253	x
"Computational Treatise [Beginning with a Problem about a] Sea Island" (*Haidao suanjing* 海島算經)	ca 263	1	9	x
"Computational Treatise of Five Departments" (*Wucao suanjing* 五曹算經)	after 386	5	67	x
"Computational Treatise of Master Sun" (*Sunzi suanjing* 孫子算經)	ca 400	2	61	x
"Computational Treatise of Zhang Qiujian" (*Zhang Qiujian suanjing* 張邱建算經)	ca 450	3	85	
"Computational Treatise on the Continuation [of Tradition] of Ancient [authors]" (*Jigu suanjing* 緝古算經)	ca 600	1	20	x
"Computational Treatise of Xiahou Yang" (*Xiahou Yang suanjing* 夏侯陽算經)	763-779	3	82	

Table 1: Date of compilation, number of chapters (limited to those containing problems), problems in each work and indication whether a punctuated edition was available on Wikisource. Title translations and dating from Keller and Volkov (2014: 62).

As is the case with many modern day math word problem datasets, the problems almost always invoke a real world context in their setting, for example, when asking about the price of purchased goods, although this does not necessarily imply that they were practical in nature (Martzloff, 2006: 54-8).

The procedures that are supplied to compute the solutions can vary in level of detail, but are in general expected to be complete in the sense that following them step by step, one is able to compute the correct answer, although substantial interference might be required in some parts. The procedures often mention the specific numbers used in the problems, and in many cases also contain intermediate numerical results.

In order to build a dataset that is useful for testing automatic solving approaches on these problems, digitized versions were sourced from Wikisource[4] and the Kanseki repository[5] (Wittern, 2016). Since the Wikisource editions have punctuation added to the texts by crowd sourced editors, they were preferred where available.

Since the division of the works into questions, answers, and procedures is expressed in the text using characteristic markers such as "suppose there is" (*jin you* 今有) in the beginning of questions, as well as layout of the text into paragraphs, it is easy to extract triplets in a semi-automated way, taking care of the occasional deviation which is to be expected in natural language documents that have been transmitted over such a long time. Commentaries contained in the texts were removed during the processing.

In total, this process resulted in 577 triplets extracted from the texts. Table 1 shows the number of problems in each work. The table also shows the number of chapters (*juan* 卷), which in mathematical books often correlate with major thematic subdivisions.

In order to use the dataset in an automatic evaluation, the answer strings of each triplet were decomposed into numerical solutions and textual elements, by searching for numerals followed optionally by a unit of measurement (UoM) with a regular expression. This step is necessary because the answers often contain additional verbiage aside from the bare result itself, for which it would be unreasonable to expect the model to correctly predict it. Cases in which a number in the answer does not represent a value to be calculated, but rather helps to structure it textually were manually fixed after-

a problem that is clearly incomplete without the general procedure. For the two general procedures for which such stub-procedures exist (the *Fangcheng* 方程 (equivalent to systems of linear equations) and *Yingbuzu* 盈不足 (double false position) procedures from the *Nine Categories*), the general procedure was appended to the stub-procedure. Furthermore, some of the works feature alternative procedures for solving the same problem. In these cases, only the first procedure given was included in the dataset.

[4] https://zh.wikisource.org/wiki/九章算術, https://zh.wikisource.org/wiki/緝古算經, https://zh.wikisource.org/wiki/海島算經, https://zh.wikisource.org/wiki/孫子算經, https://zh.wikisource.org/wiki/五曹算經 (All accessed 22.12.2024).

[5] https://www.kanripo.org/. The texts used here have numbers KR3f0038 and KR3f0039 in the database.

Question	今 有 出 錢 一萬三千五百 , 買 竹 二千三百五十 箇 。 問 : 箇 幾何 ? now have out *cash* 13 500 , buy bamboo 2350 piece . ask : piece how much ? Suppose one has paid out 13 500 *cash* [unit of currency] and purchased 2350 pieces of bamboo. Question: how much is one piece?
Answer	荅 曰 : 一 箇 , 五 錢 、 四十七 分 錢 之 三十五 。 answer say : 1 piece , 5 *cash* , 47 part *cash* genitive particle 35 . The answer says: 1 piece is 5 *cash* and 35 parts of 47 *cash*.
Converted	荅 曰 : 一 箇 , 270/47 錢 。 answer say : 1 piece , 270/47 *cash* .
Procedure	經 率 術 曰 : 以 所 買 率 為 法 , 所 出 錢 treat ratio procedure say : with relative pronoun buy ratio make divisor , relative pronoun out *cash* 數 為 實 , 實 如 法 得 一 錢 。 number make dividend , dividend match divisor obtain 1 *cash* . The procedure for treating ratios says: take the ratio of what has been bought as divisor, the number of *cash* that has been paid out as dividend, do the division, obtaining one *cash* each time the dividend matches the divisor.

Table 2: Example of a problem (number 32 from chapter 2 of the *Nine Categories*).

wards, as e.g. in cases where after buying a certain number of goods for an amount of money, the price of *one* item is sought after, and the *one* is repeated in the answer.

In the texts, most answers are in the form of quantities with an UoM attached. Often, a quantity is expressed as a compound of integers or fractions of several UoM at different levels of scale, e.g. "two *chi* [unit of length] and four *cun* [unit of length equivalent to $\frac{1}{10}$ *chi*]" (*er chi si cun* 二尺四寸). When extracting the numerical solutions, such compounds were reduced to a single rational number of the largest UoM, in the example, $2 + \frac{1}{10} \cdot 4 = \frac{12}{5}$ *chi*. The intention of this step is to align the computational steps that need to be taken to compute the result more closely to the procedures that are proscribed in the texts themselves, which in most cases tacitly assume that appropriate conversions are done by the mathematician.[6] The ability of the model to understand and potentially convert pre-modern UoM is still tested, since no conversion of any form is done for the quantities stated in the questions.

In a comparable step, mixed fractions in the answers were reduced to a single improper fraction. In general, fractions which are written in the texts using the notation "x *fen zhi* 分之 y", being equivalent to $\frac{y}{x}$ in modern notation, are always considered as a single term in the extracted answers. This runs contrary to the intent of some of the problems, notably 16 problems identified in early parts of the works, which explain basic arithmetic operations on fractions, necessitating an understanding of fractions in the answers as consisting of two numbers to be computed, numerator and denominator.[7] In most other procedures, knowledge of these operations is assumed, so modelling fractions in this way allows us to stay closer to the way procedures are written in most cases.

As opposed to most modern math word problem datasets (Kao et al., 2024), the problems extracted from the ancient Chinese mathematical texts commonly ask for the computation of several unknowns, in one extreme case even 27 (mean 2, sd 2.54). The length of the procedures associated with the problems also varies considerably, ranging (without punctuation) from 8 characters to 681 (mean 55.7, sd 64.1), indicating varying degrees of mathematical complexity of the tasks.

During manual spot-checks, some textual errors in the material were found and corrected. However, it has to be expected that the dataset is not completely free of errors.

4 Solving the Problems with a LLM

4.1 Experimental setup

In this section, the performance of a state of the art LLM, GPT-4o will be tested in solving the problems in the dataset described in the previous section. The main strategy that will be used for this is PoT (Chen et al., 2023).

[6] It should be noted however that there are some problems the main point of which are unit conversions, which are made significantly easier due to this conversion.

[7] These 16 problems are included in the dataset, but were excluded for the evaluation in Section 4.

Formally, let a problem triplet (Q_i, A_i, P_i) consist of question Q_i, answer A_i, and procedure P_i. Let $\tilde{A}_i$ be the result of decomposing A_i into textual and numerical elements as described in the previous section, containing numerical solutions $a_{i,1}, \ldots, a_{i,k}$. We then test the capability of the model to generate Python code C, such that after executing C, there are k variables a, b, … such that a has value $a_{i,1}$, b has value $a_{i,2}$ and so on.

In order to understand the impact of different factors on the performance, various prompts were tested. For the purpose of constructing these prompts in a way that avoids spoiling the result while still giving the necessary structure to give an answer, we define $\hat{A}_i$ as the string that is derived from $\tilde{A}_i$ by substituting the numerical values with letters "a", "b" and so on.

Following the by now well-known few-shot-learning approach (Brown et al., 2020), the model was presented with a series of sample inputs and expected answers. The five problems used as exemplars were chosen in a way that is reasonable for a human researcher, i.e. by pulling respectively the first problems from the first five chapters of the most famous work in the dataset, the *Nine Categories*. For each example $1 \leq j \leq 5$, aside from the solution code C_j, an English translation E_j of the problems, including the procedure, was written. One of the exemplars as well as the system prompt SP can be found in Appendix A. For reference, a zero-shot (Kojima et al., 2022) approach was also tested, the prompt for which is also in Appendix A.

Putting the elements together, the full few-shot input for the model when trying to solve problem (Q_i, A_i, P_i) consists of the sequence

$$\begin{aligned} &S(SP),\\ &U(Q_1||P_1||\hat{A}_1), M(E_1||C_1),\\ &\ldots,\\ &U(Q_5||P_5||\hat{A}_5), M(E_5||C_5),\\ &U(Q_i||P_i||\hat{A}_i), \end{aligned} \tag{1}$$

where $||$ denotes concatenation and S, U and M system, user and assistant messages respectively.

A solution generated by GPT-4o with this prompt for the problem in Table 2 is shown in Figure 2. The accuracy achieved by GPT-4o using this prompting format is shown in Table 3. The table also shows the accuracy of the zero-shot strategy, as well as several formats that are derived by modifications to (1):

```
'''Suppose there is an expenditure of
13,500 qian to buy 2,350 bamboo pieces.
Question: how much does each piece cost?
The procedure for proportional rates
says: Take the quantity bought as the
divisor, and the amount of money spent
as the dividend. Divide the dividend by
the divisor to obtain the cost per
piece.
Answer: one piece costs *a* qian.'''
# 出錢一萬三千五百 [paid out 13500 cash]
所出錢數 = 13500
# 買竹二千三百五十箇 [bought 2350 pieces
    of bamboo]
所買率 = 2350
# 以所買率為法 [take the ratio of what
    has been bought as divisor]
法 = 所買率
# 所出錢數為實 [the number of cash that
    has been paid out as dividend]
實 = 所出錢數
# 實如法得一錢 [do the division]
a = Fraction(實, 法)
```

Figure 2: Output of the model for problem 32 from chapter 2 of the *Nine Categories*, with English translation in the block comment by the model, and translations of the single-line comments in square brackets added manually afterwards. Empty lines before each single-line comment contained in the original output have been removed for better display.

Alternative system prompt For reasons that will be laid out in Section 5, the system prompt used in (1) stresses that the code should stay as close as possible to the structure of the original, and quote appropriate sections from it in comments before each block of code. To test whether this prevents the model from adopting more direct solutions, an alternative system prompt without these restrictions was tried (see Appendix A for the prompt).

No punctuation For this configuration, punctuation was removed from all strings in the input. It is important to test the ability of the model to cope with text written without punctuation, as this is the format that the works were transmitted to us, and modern punctuations are not available for all texts. The results for the texts from the dataset without punctuation, for which this was the only configuration tested, are shown in Table 4.

No translation In this format, the English translations $E_1, \ldots, E_5$ were removed from the prompt. Doing a complete translation first would be reasonable approach for a human tasked with solving the problems, so we test whether this also helps the model achieve higher accuracy.

Method	All		*Nine Categories*		*Sea Island*		*Five Departments*		*Master Sun*		*Continuation*	
	Mean	**B-o-5**	**Mean**	**B-o-5**	**Mean**	**B-o-5**	**Mean**	**B-o-5**	**Mean**	**B-o-5**	**Mean**	**B-o-5**
Zero-shot	44.0	60.7	40.8	61.0	4.4	11.1	60.6	70.1	58.9	77.2	0.0	0.0
Few-shot (default)	51.1	63.0	53.0	66.1	13.3	33.3	52.5	65.7	65.6	73.7	0.0	0.0
Few-shot (alternative prompt)	52.3	62.0	54.1	64.8	17.8	33.3	56.4	62.7	64.2	75.4	0.0	0.0
No punctuation	47.8	60.7	46.6	61.0	6.7	33.3	58.2	68.7	63.9	75.4	0.0	0.0
No translation	51.1	59.4	52.6	61.0	13.3	22.2	55.2	64.2	63.5	73.7	0.0	0.0
No procedures	36.7	45.5	33.8	42.8	2.2	11.1	47.2	53.7	54.0	66.7	1.0	5.0
Numerical solutions provided	54.2	67.6	57.7	72.9	4.4	11.1	58.5	68.7	61.1	73.7	2.0	10.0

Table 3: Accuracies of different prompting strategies on the punctuated dataset and by title in percent. Values in the mean columns are averaged over five runs of the model, and in the best-of-5 (B-o-5) columns a problem is counted as solved if it was solved in at least one run.

All		*Xiahou Yang*		*Zhang Qiujian*	
Mean	**B-o-5**	**Mean**	**B-o-5**	**Mean**	**B-o-5**
37.4	49.7	45.6	57.3	29.4	42.4

Table 4: Accuracies in percent for the titles in the dataset where no punctuation was available

No procedures In order to test whether the information provided in the question alone is sufficient for the model to derive a solution, the procedures $P_1, \ldots, P_5$ and P_i are removed from the prompt. Furthermore, the Python solutions $C_1, \ldots, C_5$ were streamlined to not include steps described in the procedure but unnecessary in Python, and comments that quote the procedures are changed into English comments that explain the reasoning. The system prompt was also changed by removing the instruction to quote the procedure before each block of code. This brings the format much closer to conventional math word problem setups, which usually do not contain procedures for computing results.

Numerical solutions provided In this format, the solutions $\hat{A}_1, \ldots, \hat{A}_5$ and $\hat{A}_i$ are modified by adding the values of the unknowns as Arabic numerals, giving the model an opportunity to cheat by knowing the correct solutions.

4.2 Discussion

As can be seen in Table 3, the performance in all of the works leaves much room for improvement. Unsurprisingly the setup where the model has access to the solutions it is supposed to compute shows the best accuracy, although even in

Category	Type	Count
global	misunderstood procedure	7
	misunderstood procedure (inference required)	5
	misunderstood question	4
localized	misunderstood expression	27
	unit conversion	17
	code error (fractions)	16
	math reasoning	4
	textual error	3
	code error (variable name)	2
	code error (syntax)	1
	result rounded in the text	1

Table 5: Count of errors by type encountered during the evaluation of 50 randomly chosen failed solutions

that scenario, mean accuracy is significantly lower than that reported for the same model on the MATH dataset, 68.5%[8], which was designed to be challenging for expert humans (Hendrycks et al., 2021). While removing procedures outright leads to a large drop in accuracy, telling the model to focus on staying close to the procedure (default prompt) or not (alternative prompt) does not cause a statistically significant difference. Removing punctuation leads to a small but significant drop in mean accuracy. Removing translations did not have a significant effect on mean accuracy.

Looking at the breakdown of the results by title, we can observe considerable differences be-

[8] https://github.com/openai/simple-evals (entry gpt-4o-2024-11-20, accessed 23.01.2025).

tween the accuracy values. In particular, the model was almost unable to solve any problems in the *Sea Island* and the *Continuation*. Inside a single work, the performance can also vary considerably by chapter. For example, the mean accuracy using the default few-shot prompt for the worst performing chapter of the *Nine Categories* is as low as 21%.[9] Since chapters often group thematically related problems, this indicates that the model had more difficulties in solving certain types of problems.[10]

Running a logistic regression shows that both the total textual length of a problem, i.e. the sum of the lengths in characters without punctuation of question, answer, and procedure, and the number of unknowns in it are significant predictors on the model being able to solve it.[11] As a case in point, both of the two works with the worst performances each contain problems or procedures that are much longer than the average of the dataset. The mean number of characters (not counting punctuation) per problem is 164 for the *Sea Island* and 311 for the *Continuation*, but only 106 for the *Zhang Qiujian*, which has the third highest values in this regard. In the *Continuation*, many problems are further complicated by a high number of unknowns to be computed, 6.95 on average per problem, compared to 1.99 for the text with the second highest value. At the same time, it can be considered more advanced in terms of computations involved, as almost all of the problems require the extraction of cubic roots (Lim and Wagner, 2017: 27). Accordingly, in its context as a historic textbook, it was the only one of the works considered here reserved for a program for advanced students (Keller and Volkov, 2014: 61).

In order to gain a deeper understanding of why the model fails to output correct code, a manual error analysis was conducted. 50 problems were randomly sampled from among those where the few-shot prompting strategy failed in all five runs. For each of those, the output for one of these runs was then annotated, by first determining whether the code was in general following the structure of a correct solution and could be fixed by modifying a few localized sections, or whether it was completely unusable, because either the model did not understand the the structure of the procedure provided, or the intent of the question. Table 5 gives an overview of the errors encountered.

In the cases where the code was fixable with localised modifications, the type of error in these locations was further analysed. Of course, there are many possibilities for generating incorrect results. However, a few major categories can be clearly distinguished.[12]

First, there were 27 cases where the model misunderstood an expression in the original text, e.g. translating the expression "*tai ban sheng* 太半升" (two-thirds of a *sheng*) into "half a sheng" in the English translation and "$\mathrm{Fraction}(1,\ 2)$" in the code section of the output.

Second, there were 17 cases where the model made an error in doing unit conversions, reflecting either a lack of world knowledge or applied mathematical reasoning skill.

Third, there were 19 cases where, judging from the translation and the code it has produced, it intended to do the right thing, but failed to produce working code. 16 of these are related to our choice to force the model to use fractions in computing results, running counter to the semantics of Python defaulting to floating point numbers when doing divisions with "/" or taking square root with "$\mathrm{math.sqrt}$". To our surprise, in two of the examined outputs, the model generated code that was invalid because of wrong variable names, in both cases because it had used the same name with traditional characters in one place in the code, and then with simplified characters in another location. In one case, the syntax of the generated code was incorrect.

Third, there were four cases where the problem seems to have been with the mathematical reasoning of the model. While the procedures supplied with the problems in most cases give a complete solution strategy, they are often not detailed to the level that they could simply be mindlessly followed, and not all mathematics can be left to the

[9] The complete table showing accuracy for each chapter can be found in Appendix B.

[10] As groups of more specifically related problems also tend to cluster inside each chapter, we would have furthermore expected to encounter clusters of easier or more difficult problems when arranging them by their position inside each chapter. However, a runs test only gave a significant result for one single chapter, number 2 in the *Nine Categories*, which is clearly divided into two distinct parts, with the first part containing a family of problems that is much easier than those in the second part.

[11] In the few-shot configuration using best-of-5 for evaluation. $\beta_{text\_len} = -0.0046629$, $SE_{text\_len} = 0.0009004$, $p_{text\_len} < 0.001$, $\beta_{n\_unknown} = -0.3759104$, $SE_{n\_unknown} = 0.0562560$, $p_{n\_unknown} < 0.001$.

[12] Examples of the most commonly encountered categories can be found in Appendix C.

Python interpreter. For example, there was a case where the procedure simply stated to add a "difference", which in the context could refer to two quantities, of which the model chose the wrong one.

Finally, three cases were caused by an error in the original text, and one by the solution provided in the original being rounded, with no indication in the procedure that such a step has to be taken.

5 Further Use of the Generated Code

In the previous section, the dataset was used as a benchmark to test a model's ability to output code to calculate correct solutions for the problems. However, as mentioned in Section 1, for the historical problems used here, this may not be the most relevant task. In this section, it will be shown that the code that is generated to compute solutions is interesting in its own right, because it allows us to explore ancient Chinese mathematics from a perspective that was much more difficult to attain before: what were the calculations needed to solve the problems?

Of course, using the digitized full-text editions of the texts, it is trivial to search the procedures for the presence of words that commonly proscribe certain mathematical operations, e.g. "*cheng* 乘" (to multiply). However, this does not necessarily give us a complete picture: On the one hand, there are some polysemous lexemes that can signify different operations, such as "*chu* 除" (to divide, to subtract), or an operation and something entirely else, e.g. "*cong* 從" (length, to accord to, to add). On the other hand, there can be operations that are implicit, and not overtly expressed in the text. For example, as described in Section 3, unit conversions are often left for the practitioner to fill in, and might, due to the system of UoMs used, require non-trivial multiplications or divisions.

By including the instruction for the model to closely follow the structure of the provided procedure, and quote the relevant section of the procedure before each section of code (see Appendix A for the prompt), we ensure that code can be aligned to the original text. While this does of course not ensure a perfect match between the Python implementation and how a practitioner would have done their calculations, the alignment allows a targeted search for the code equivalents of expressions. Furthermore, by restricting ourselves to problems where the Python solution computes the correct answers, without having them spoiled in the prompt, we can be confident that the model understood the problem at least to the level that it could independently solve it.

Among the correct outputs produced by the default few-shot prompt, 76.1% of code blocks divided by empty lines produced by the model were started by a comment that could be matched to a portion of either the question or procedure provided in the prompt. In order to check how reliable this alignment is, a manual analysis of 50 randomly sampled solutions was conducted. In particular, it was examined whether 1) the mathematical operations specified in the text quoted as a comment before each block match the semantics of the code in the block 2) there are calculations specified in the procedure missing from the code 3) there are calculations in the code proscribed neither explicitly nor implicitly (e.g. UoM conversions) in the procedure. Code blocks that contain no calculations but just assign variables were not checked.

In 43 of the 50 cases, no problems were discovered according to the three criteria. In two cases, both from the *Fangcheng* (equivalent to Gaussian elimination) section of the *Nine Categories*, the procedure given contains specific instructions for one paradigmatic problem, which need to be generalized to the problem at hand.[13] Hence, it is impossible to produce code that aligns perfectly to the instructions in the procedure. In two cases, the procedures contained steps needed to deal with fractions, which were rendered unnecessary by using the `Fraction` class and thus not included in the generated code. In one case each, the comments were translated into English, an algorithm that completely deviated from the procedure was adopted by the model, and one of the code blocks contained code that did not match the quoted section from the procedure.

To demonstrate the potential of this alignment between original text and its translation into code, blocks were searched for multiplication with the operator *, and the accompanying comments were then analysed for the presence of several constructions that commonly denote multiplication. The results, grouped by title, are displayed in Table 6. The table also contains a column for UoM conversions, which are often not explicated in the original text, but for which the model almost always added a separate comment explaining the step. As can

[13] See Footnote 3 above.

Title	*cheng* 乘	*n zhi* 之	*bei* 倍	*ming* 命	UoM conversion	other
Nine Categories	141 (56%)	29 (12%)	10 (4%)	0 (0%)	54 (21%)	20 (8%)
Five Departments	43 (90%)	0 (0%)	0 (0%)	0 (0%)	4 (8%)	2 (4%)
Master Sun	46 (68%)	1 (1%)	4 (6%)	3 (4%)	13 (19%)	5 (7%)
Sea Island	5 (62%)	0 (0%)	0 (0%)	0 (0%)	3 (38%)	0 (0%)

Table 6: Constructions in sections of the procedures quoted before code blocks containing multiplications (* in Python). Multiple categories can apply for the same code block.

be seen, "*cheng* 乘" (to multiply), is the most frequently used lexeme to express multiplication. All of the texts also employ other ways for stating this operation. However, the popularity of these differs significantly between the works, with "n *zhi* 之" ("to n it", where n is a natural number) being the most frequent in the earliest text in the dataset, the *Nine Categories*, and seemingly dropping more or less out of fashion in the later texts.

In the current setup, a major drawback is of course that it is limited to those problems that the model has successfully solved. As we have seen, misunderstanding of the text is one of the main reasons for errors in the output, and lexemes that are infrequently used in a certain meaning might be one of the main causes for misunderstandings. For example, "*ming* 命" (to command, to name, to multiply) is used to refer to multiplication in a few procedures in the *Nine Categories* (Chemla and Guo Shuchun, 2004: 963-4), but the model failed to produce correct solutions for these. Hence, at the current stage, the setup is mostly suited to discover larger trends. However, we are confident that with simple means, the accuracy can be further improved. Possible directions for this will be discussed in the next section.

6 Conclusions

In this article, we have introduced a dataset of ancient Chinese math word problems, and established a baseline performance for an LLM solving the problems using a PoT approach. While the fact that around two thirds of the problems could be solved in this setting shows the general potential of using LLMs for historic mathematics, it of course leaves much room to improvement. By releasing the dataset, we hope to encourage further research in this direction.

In particular, an obvious first step would be to explore recent advancements over the basic PoT prompting used here, by e.g. giving the model feedback on its solution attempts (Zhou et al., 2023). Of course, more sophisticated models which achieve higher scores in modern day math benchmarks could also be tried. However, as our error analysis in Section 4.2 has revealed, the biggest challenge for GPT-4o does not appear to be in mathematical reasoning, but rather in understanding the language of the texts. In this regard, a fruitful approach to explore could be in using retrieval augmented generation, by giving the model explanations of technical terms contained in the problem. Resources that could be used for this purpose include both pre-modern commentaries that accompany several of the texts in the dataset, as well as modern day glossaries compiled by historians.

A limitation that might require more fundamental changes to the setup is the model of computation used. As outlined in Section 3, the choice taken here was motivated by the consideration to have the code output correspond as directly as possible to the procedures provided in the texts. However, using Python with rational numbers represented by the `Fraction` class is clearly not ideal. First, it does not align perfectly which what is explicit and implicit in the procedures. Second, as we have seen in Section 4.2, GPT-4o sometimes had considerable problems in producing working code under these conditions. Overcoming these could potentially entail designing a simple custom programming language as the target for translations, or at least providing a set of purpose-built library function that match the semantics of certain compound operations in ancient Chinese mathematics, e.g. root extraction or operations with mixed fractions.

As the sketch of future applications in Section 5 demonstrates, doing further research in this direction might prove very fruitful, because it allows us to approach historical mathematics from a new perspective, guided by a large scale analysis of the actual computations performed, without the need for time-consuming human annotation.

Limitations

The text in the dataset were not systematically checked to ensure that they are free from errors.

Only a single model and a single set of exemplars was tested.

No comparison was made against human performance on the dataset.

Acknowledgements

I am grateful for the financial support from the Alexander-von-Humboldt foundation, from a grant awarded to Andrea Bréard. I would like to thank Andrea Bréard and three anonymous reviewers for their helpful remarks on the article.

References

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large Language Models for Mathematical Reasoning: Progresses and Challenges. *Preprint*, arXiv:2402.00157.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, and Amanda Askell. 2020. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

Karine Chemla and Guo Shuchun. 2004. *Les Neuf Chapitres: Le classique mathématique de la Chine ancienne et ses commentaires*. Dunod, Paris.

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

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

Joy He-Yueya, Gabriel Poesia, Rose E. Wang, and Noah D. Goodman. 2023. Solving Math Word Problems by Combining Language Models With Symbolic Solvers. *Preprint*, arXiv:2304.09102.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. *Preprint*, arXiv:2103.03874.

Kuei-Chun Kao, Ruochen Wang, and Cho-Jui Hsieh. 2024. Solving for X and Beyond: Can Large Language Models Solve Complex Math Problems with More-Than-Two Unknowns? *Preprint*, arXiv:2407.05134.

Agathe Keller and Alexei Volkov. 2014. Mathematics Education in Oriental Antiquity and Middle Ages. In Alexander Karp and Gert Schubring, editors, *Handbook on the History of Mathematics Education*, pages 55–83. Springer, New York.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. *Advances in Neural Information Processing Systems*, 35:22199–22213.

Tina Su-lyn Lim and Donald B. Wagner. 2017. *The Continuation of Ancient Mathematics: Wang Xiaotong's "Jigu Suanjing", Algebra and Geometry in 7th-Century China*. NIAS Press, Copenhagen.

Jean-Claude Martzloff. 2006. *A History of Chinese Mathematics*, 2nd edition. Springer, Berlin.

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

Christian Wittern. 2016. Kanseki Repository. *CIEAS Research Report 2015*, Special issue:1–80.

Wu Wen-Tsun. 2019. 吳文俊全集 - 數學史卷 *[The Complete Works of Wu Wen-Tsun - History of Mathematics]*. Kexue chubanshe, Beijing.

Wu Wenjun and Shen Kangshen, editors. 2000. 中國數學史大系第四卷西晉至五代 *[Compendium of the History of Chinese Mathematics Volume 4: From Western Jin to the Five Dynasties]*. Beijing shifan daxue chubanshe, Beijing.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2023. Solving Challenging Math Word Problems Using GPT-4 Code Interpreter with Code-based Self-Verification. *Preprint*, arXiv:2308.07921.

A System prompts and exemplar

Default system prompt: "Translate ancient Chinese math problems into Python code, ensuring that each section of code adheres to the structure of the procedure ('術') provided. Put the part of the procedure that corresponds to each block of code as a comment before the block. Ensure that the complete procedure is encoded. Use the class 'Fraction' to represent numbers that might not be integers and use the appropriate units for each calculation. Don't use any other external functions."

Alternative system prompt: "Translate ancient Chinese math problems into Python code. Ensure that the complete procedure is encoded. Use the class 'Fraction' to represent numbers that might not be integers and use the appropriate units for each calculation."

System prompt for scenario without procedures: "Translate ancient Chinese math problems into Python code. Use the class 'Fraction' to represent numbers that might not be integers and use the appropriate units for each calculation. Don't use any other external functions."

Zero-shot system prompt: "Translate ancient Chinese math problems into Python code that computes the values of the unknowns replaced by letters ('a', ...) in the answer ('荅' or '答'). Write the solutions into variables that have exactly the same name as the unknowns. Use the class 'Fraction' to represent numbers that might not be integers and use the appropriate units for each calculation. Don't use any other external functions."

Exemplar for problem 1 from chapter 1 of the *Nine Categories*:

User: "今有田廣十五步，從十六步。問：為田幾何？
方田術曰：廣從步數相乘得積步。以畝法二百四十步除之，即畝數。百畝為一頃。
荅曰：a 畝。"

Assistant:

```
'''
Suppose there is a field with a
    ↪ width of 15 bu and a length
    ↪  of 16 bu.
Question: how large of a field
    ↪ does it make?

The procedure for rectangular
    ↪ fields says: The numbers of
    ↪  bu in width and length are
    ↪  multiplied with each other
    ↪ , obtaining the [number of]
    ↪  accumulated bu.
When dividing it by the mu-
    ↪ divisor, 240, it is the
    ↪ number of mu.
100 mu make 1 qing.

The answer says: *a* mu.
'''

#廣十五步
廣步數 = 15

#從十六步
從步數 = 16

#廣從步數相乘得積步
積步 = 廣步數 * 從步數

#畝法二百四十步
畝法 = 240

#以畝法二百四十步除之，即畝數
畝數 = Fraction(積步, 畝法)

a = 畝數
```

B Results by chapter

Table 7 shows accuracy separately for each chapter.

C Examples of errors

Due to space considerations, only examples of the most frequent categories of localized errors are reproduced here. The complete annotated files can be found at https://github.com/notiho/ancient-chinese-math-problems.

Misunderstood expression (Problem 17 from chapter 1 of the *Nine Categories*): The relevant fragment in the original text is "又有三人，三分人之一，分六錢三分錢之一，四分錢之三" (Again, there are three and one third of a person, dividing six *cash*, one third *cash* and three fourth *cash*). The model translates this as "Suppose there are three people. Each person is to receive one-third of the total. The total is 6 qian, plus one-third of a qian, plus three-fourths of a qian." and produces the following code:

```
# 三人
人數 = 3
```

Expected:

```
人數 = 3 + Fraction(1, 3)
```

Unit conversion (Problem 21 from chapter 3 of the *Master Sun*): LLM output:

```
# Convert sheng to hu (10 sheng =
    ↪  1 hu)
a = Fraction(總食量升, 10)  #
    ↪ Total food in hu
```

Expected (100 *sheng* = 1 *hu*):

Title	Chapter	Problems	Zero-shot		Few-shot		Few-shot (alt. prompt)		No punctuation		No translation		No procedures		Numerical solutions provided	
Nine Categories	1	25	56.0	72.0	83.2	92.0	88.8	92.0	74.4	84.0	92.0	92.0	49.6	56.0	94.4	96.0
	2	45	24.4	46.7	52.4	64.4	50.2	57.8	40.9	53.3	57.3	62.2	7.1	8.9	56.0	73.3
	3	19	51.6	63.2	65.3	78.9	66.3	78.9	61.1	73.7	64.2	68.4	54.7	57.9	62.1	78.9
	4	23	49.6	65.2	59.1	69.6	62.6	82.6	54.8	73.9	52.2	73.9	53.0	56.5	68.7	91.3
	5	34	37.6	64.7	58.8	70.6	56.5	64.7	56.5	64.7	49.4	55.9	34.1	41.2	63.5	73.5
	6	28	40.7	60.7	41.4	50.0	37.9	46.4	38.6	53.6	45.7	50.0	25.7	39.3	43.6	50.0
	7	20	26.0	45.0	21.0	35.0	23.0	35.0	11.0	30.0	16.0	30.0	52.0	60.0	23.0	50.0
	8	18	53.3	66.7	40.0	55.6	40.0	61.1	35.6	55.6	46.7	72.2	45.6	72.2	50.0	77.8
	9	24	46.7	75.0	48.3	75.0	59.2	70.8	42.5	62.5	41.7	45.8	17.5	37.5	51.7	66.7
Five Departments	1	19	68.4	73.7	34.7	63.2	46.3	63.2	49.5	73.7	25.3	36.8	56.8	63.2	34.7	52.6
	2	12	68.3	75.0	63.3	66.7	66.7	66.7	65.0	66.7	81.7	91.7	51.7	58.3	65.0	66.7
	3	14	84.3	92.9	78.6	85.7	85.7	85.7	81.4	85.7	84.3	85.7	55.7	57.1	87.1	92.9
	4	12	21.7	41.7	40.0	50.0	33.3	41.7	36.7	50.0	45.0	58.3	16.7	25.0	38.3	50.0
	5	10	50.0	60.0	52.0	60.0	50.0	50.0	60.0	60.0	52.0	60.0	48.0	60.0	80.0	90.0
Master Sun	2	23	47.0	78.3	60.9	73.9	57.4	73.9	54.8	73.9	56.5	73.9	45.2	56.5	53.0	69.6
	3	34	67.1	76.5	68.8	73.5	68.8	76.5	70.0	76.5	68.2	73.5	60.0	73.5	66.5	76.5
Sea Island	1	9	4.4	11.1	13.3	33.3	17.8	33.3	6.7	33.3	13.3	22.2	2.2	11.1	4.4	11.1
Continuation	1	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	5.0	2.0	10.0

Table 7: Accuracy in percent by chapter

```
a = Fraction(總食量升, 100)
```

Code error (fractions) (Problem 8 from chapter 1 of the *Five Departments*): LLM output:

```
# 以四除之
邊長 = 方周 / 4
```

Expected (division produces fractional result):

```
邊長 = Fraction(方周, 4)
```