GENERATING CODE TO VERIFY CRYPTIC CROSSWORD REASONING

Martin Andrews*
Red Dragon AI, Singapore
martin@reddragon.ai

Sam Witteveen
Red Dragon AI, Singapore
sam@reddragon.ai

ABSTRACT

Cryptic crossword clues are challenging linguistic reasoning puzzles, with fresh, real-world test cases published daily in major newspapers globally. Unlike standard crosswords, cryptic clues uniquely combine a 'definition' with intricate 'wordplay' – a logical puzzle embedded in language – designed to prove each answer's correctness through reasoning alone, independent of grid context. This work describes an open-licensed, LLM-driven system for automated cryptic crossword solving which generates: (i) answer hypotheses and wordplay explanations; and (ii) code-based verification of the proposed reasoning, ensuring solution validity. Critically, our system achieves state-of-the-art performance on the challenging Cryptonite dataset, comprising cryptic clues from prestigious UK newspapers like The Times and The Telegraph. Furthermore, by expressing each verified solution and its reasoning in executable Python code, our system offers unprecedented interpretability, allowing for detailed inspection and analysis of the automated cryptic crossword solving process.

1 INTRODUCTION

Robust verification is increasingly recognized as essential for reliable reasoning in the fields of mathematics (Jiang et al., 2023; Yang et al., 2023; Trinh et al., 2024) and code generation (Ni et al., 2023; Ridnik et al., 2024). Applying formal verification to *linguistic reasoning*, however, remains less explored. This work addresses the compelling reasoning challenge of Cryptic Crossword solving, which provides a valuable test-bed for advancing reasoning capabilities, and has these key attributes:

- Cryptic crosswords are a globally popular intellectual challenge. Solving them requires sophisticated NLU, logical inference, and contextual nuance – core human reasoning skills. This makes them a rigorous, human-relevant benchmark for LLM reasoning evaluation.
- Decades of puzzles from major newspapers offer a vast and growing resource for training and testing (contrasting with IMO/AIME problems, which have a much lower number of novel problems).
- By construction, Cryptic clues have one 'true' reasoning path, with the wordplay acting as a self-contained proof, ensuring each solution's validity independent of external context. This wordplay, however, is not merely an explanation but a linguistic puzzle in its own right, demanding ingenuity to see how its seemingly disparate pieces interlock.

In this work, taking the cue from the effectiveness of provers coupled with verifiers for mathematical reasoning tasks (Jiang et al., 2023), we tackle cryptic crosswords with an LLM-driven system that: (i) generates `answer` candidates; (ii) creates informal proofs (i.e. coming up with `wordplay`); and (iii) performs formalisation (which rewrites the `wordplay` logic in Python). This code-based verification offers a novel approach to evaluating LLM reasoning in complex linguistic tasks.

1.1 AN EXAMPLE CRYPTIC CLUE

As a concrete example, and to clarify the terminology used, consider the following *moderately* complex cryptic clue[1] : "Cut up over politician on the French case (7)". Solvers must parse the clue

*Corresponding Author.

[1]The Times UK (6-March-2024), Cryptic #28857 clue 4D

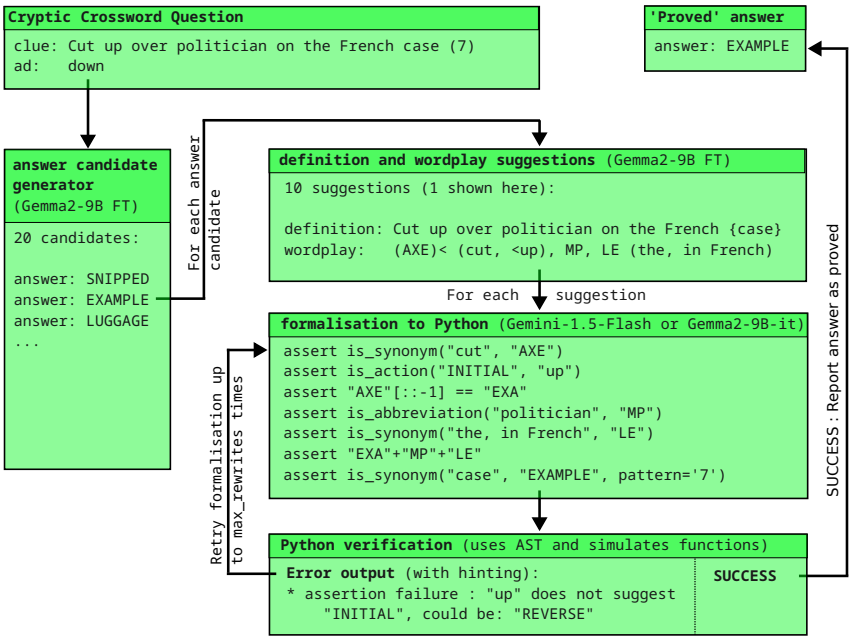

Figure 1: Proving process: `answer` candidate → `wordplay` → LLM formalisation

carefully to separate the `definition` (which acts as a regular crossword clue) and the supporting `wordplay` that can be used to arrive at the same `answer` from two directions. Arriving at the same answer by two paths constitutes the necessary proof that the correct answer has been found. See Figure 2a for a visual depiction of the reasoning involved.

1.2 Contributions

The following are the main contributions of this work:

- **An open-license system for reasoning over Cryptic clues** - our pipeline (illustrated in Figure 1) enables 9B-scale local models to achieve state-of-the-art results on the Cryptonite dataset.
- **Local models for cryptic clue tasks** - We show how local LLMs can be fine-tuned to produce `answer` candidates, and `wordplay` suggestions, and then prompted to perform Wordplay formalisation. Following an approach akin to mathematical statement formalisation, but where there are *less than 10* examples of 'good proofs' available, our novel pipeline was specifically engineered to avoid 'reasoning steps' becoming stuck in dead ends.
- **Python domain-specific verifier** - Using the output of the formaliser, the verifier presented here deconstructs the Python AST, so that it can evaluate each `assert` statement on a line-by-line basis. We believe that this is somewhat novel, since it enables the verifier to not only indicate whether the proof is valid overall, but also point to specific failures (used to regenerate failed formalisations) on all proof lines simultaneously.

To promote further study in this area, all code for the system is made publicly available at `anon`.

2 Related Work

2.1 Regular Crosswords

Non-cryptic ("regular") crosswords are known throughout the world, and are the predominant type found in newspapers in the U.S.A. One key difference from cryptic crosswords is that individual regular crossword clues are generally not 'standalone' - there may be a number of different answers that fit the given clue. The key to solving regular crosswords is thus the interaction between answers (i.e. the crossing-words), which allows for planning/backtracking to enable solving rates in the high 90% range (Wallace et al., 2022).

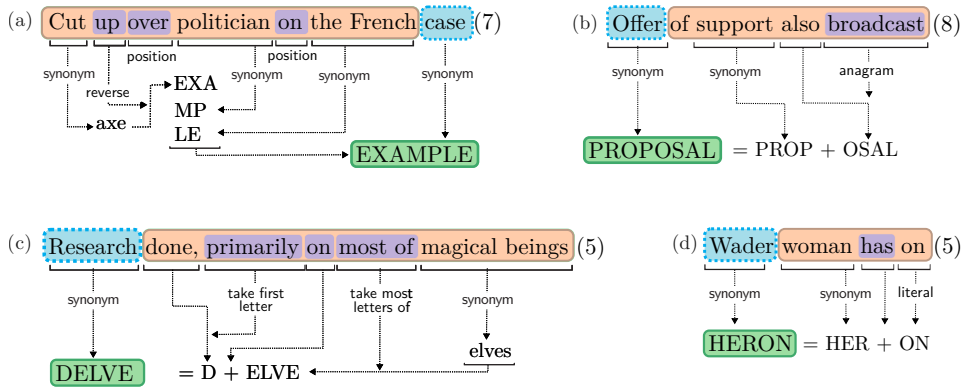

Figure 2: Clue solving illustrations. Answers are in green, definitions in blue (dashed frame), wordplays in orange, and indicators in purple. Further textual examples can be found in Appendix A.1

This work, in contrast, focuses on the solving of clues on a standalone basis, which requires elements of reasoning through the wordplay present in cryptic clues.

2.2 Cryptic Crosswords

In an 800 participant research study into the backgrounds of cryptic crossword solvers (Friedlander & Fine, 2016), the following observation was made about the skills required to solve these linguistic/reasoning puzzles:

> "... cryptic crossword skill therefore appears to be bound up with code-cracking and problem-solving skills of a quasi-algebraic nature. Conversely, lexical ability, although no doubt valuable, does not appear to be a critical discriminator of high expertise among elite solvers."

Cryptic crosswords have received surprisingly little attention from the machine learning community, despite being a notable language-oriented reasoning puzzle with global appeal. One possible reason is that cryptic crosswords are much less common in the United States than 'regular crosswords'. See Webb (2024) for an inspiring demonstration of expert solving a cryptic crossword in real-time.

The benchmark dataset used by this work is Cryptonite (Efrat et al., 2021) - a large-scale dataset of Cryptic Crossword clues from The Times and The Telegraph (major UK newspapers). The dataset contains 523,000 naturally sourced clues (published between 2001 and 2020), with the train, validation and testing splits being chosen so that a given `answer` can only appear in one of the splits.

2.2.1 Rule-based solvers

Williams & Woodhead (1979) is an early example of attempting to devise a formal language for describing cryptic clues. However, the linguistic elements of the clues tend to thwart a strictly formal approach.

A more flexible rule-based solver with a manually-crafted probabilistic grammar was introduced in Deits (2015; 2022). Building on the assumption that a `clue` can usually be split into `wordplay` and `definition`, the solver tries to find the most probable parse such that the `wordplay` yields a semantically-similar result to the `definition`. The logical form of this DSL approach is very appealing. However, it appears limited to solving clues where the `wordplay` is somewhat simple (due to the combinatorial explosion of possibilities created by longer/more complex clues).

The goal of this work is to use the flexibility of LLMs to enable a far wider range of clues to be attempted, with the aid of a formaliser/verifier to check the solutions.

2.2.2 LLM-BASED SOLVERS

Cryptonite is a challenging task for LLMs : Efrat et al. (2021) reported that fine-tuning T5-Large (a 770M encoder-decoder model) on Cryptonite's 470k cryptic clue training set achieved only 7.6% test set accuracy, slightly below the 8.6% accuracy of the rule-based clue solver of Deits (2022). Interestingly, prior to 2024, even large-scale Language Models scored very poorly on cryptic clues, likely due to (i) the misleading surface reading of the clues; (ii) the obliqueness of the definitions; and (iii) the reasoning steps required to *prove* the answer correct based on the wordplay.

Recent works, such as Sadallah et al. (2024) and Saha et al. (2024), tackle cryptic crosswords with more up-to-date local models, and commercial LLMs. Saha et al. (2024) reports results with 5- and 10-Shot prompting (without fine-tuning models), but also includes a wide-ranging study of the capabilities of models for crosswords in general.

In this work, we use a pipeline of 9B-scale LMs to produce `answer` candidates and `wordplay` suggestions, followed by a third LM to formalise each proposed solution using Python code and then rewrite/update the solutions based on feedback from a purpose-built verifier.

2.3 CODE & REASONING

To compensate for LLM *approximate generation* of logical reasoning, techniques like PAL (Gao et al., 2023) exploit LLMs' facility for writing code to create verifiable reasoning chains. An important influence on this work was also the Draft, Sketch, and Prove framework (Jiang et al., 2023) which uses an LLM to draft and create proofs that are then verified formally.

Informed by the evolution from AlphaCode (Li et al., 2022), in which huge numbers of programs were generated and filtered in order to generate valid solutions, to AlphaCodium (Ridnik et al., 2024), in which solutions were iterated upon and required much less computation, this work uses a verifier that can feed back 'hints' to the formalising LLM, so that the task of re-writing nearly-valid proofs is made easier.

2.4 WORDPLAY DATASET

The Wordplay dataset (Andrews, 2024) - an example from which is given Appendix A.4.2 - consists of data gathered from websites where cryptic crossword enthusiasts post solutions on a daily basis for each of the major publications. Each completed puzzle is annotated by an individualsolver that lists the approximately 20 clues with their `definition`, `wordplay` and `answer` fields. Note that each solver can chose their own 'standard' for writing out the `wordplay`, leading to a significant variation in `wordplay` annotation styles (even across time for an individual solver).

The Wordplay dataset deliberately follows the train, validation, and test splits defined by Cryptonite.

3 METHODS

The overall system described in the work is illustrated in Figure 1. The order of operations for the pipeline was chosen based on watching human solvers - who report going through the following steps: (a) attempting to parse the clue in a number of ways, trying to isolate the definition from the wordplay; (b) seeing which parts of the wordplay they are most confident about; (c) 'having a hunch' about the final answer; and (d) gaining a full understanding of how a clue's `wordplay` works (such that the function of every element can be explained) as proof of the overall process.

Observations of the behaviour of GPT-4 using Chain-of-Thought prompts Wei et al. (2023) (or more recently 'o1'), suggest that even very capable models tend to fixate early on during the reasoning process, and are only rarely able of completely re-hypothesising.

3.1 CANDIDATE `answer` GENERATION

Our first step to solving a given cryptic clue is to generate multiple `answer` candidates from the original `clue`, `pattern` and `ad` (across/down) fields. For this task, we fine-tuned a Gemma2 9B base model (Gemma Team & Google DeepMind, 2024) using the LoRA (Hu et al., 2021) implemen-

Gemma2 answer candidate

```
def proof(answer="DELVE",
          clue="research done, primarily on most of magical beings",
          pattern='5'):
```

Gemma2 wordplay

```
  """
  definition: research done, primarily, on most of magical beings
  wordplay: D[one] (primarily) ELVE[s] (magical beings, most of)
  """
```

LLM formalisation

```
  assert action_type("primarily", Action.INITIALS)
  assert "DONE"[:1] == "D"
  assert is_synonym("magical beings", "ELVES")
  assert action_type("most of", Action.REMOVE_LAST)
  assert "ELVES"[:-1] == "ELVE"
  assert "D"+"ELVE" == "DELVE"
  assert is_synonym("research", "DELVE", pattern='5')
proof()
```

Figure 3: Python proving: `answer` candidate → `wordplay` → LLM formalisation

tation provided by the `unsloth` package (unsloth.ai, 2024). The model was trained for 1 epoch on the Cryptonite training set of approximately 470,000 examples.

For each `clue` being evaluated, we generate 20 valid `answer` candidates, where candidates that did not match the `pattern` were immediately rejected and regenerated, and those not contained in the crossword words list (Beresford, 2000) were filtered out[2]. The number of candidates was chosen to balance generation cost with likelihood of the correct answer appearing in the candidate list.

3.2 Generation of `definition` and `wordplay` suggestions

To train the `wordplay` suggestion model, which translates each `answer` candidate into multiple `definition` and `wordplay` suggestions, we make use of the Wordplay dataset of Andrews (2024). For this task, we fine-tuned another Gemma2 9B base model using LoRA. The model was trained on 4 epochs on a set of approximately 16,800 examples (consisting of solution explanations of puzzles from The Times and The Financial Times from selected authors in the Wordplay dataset).

3.3 Python formalisation

Rather than create a dataset with many examples of formalisation, here we use in-context prompting with less than 10 examples of the formalisation style required. In preliminary work, we concluded that the available Gemini-Flash LLM was not capable of using a (novel) cryptic crossword domain specific language ("DSL") through in-context learning with so few examples. In contrast, we found that the LLM could be prompted to produce Python code with relative ease, so the approach taken was to frame a declarative-style-DSL as Python function calls within `assert` statements. The LLM was found to be able to reliably produce syntactically correct Python, and use the 'external functions' that had been described (as illustrated in Figure 4) to form logical sequences of declarations, which could then be parsed line-by-line by manipulating the Python abstract syntax tree ("AST"). An example of the Python DSL being generated by the formalisation LLM is given in Figure 3, with the workings of the clue solution being illustrated in Figure 2c.

To formalise `wordplay` into Python 'proofs' of the correctness of solutions, we used Google's `Gemini-Flash-1.5-001` LLM (a pinned model version) during development. This model was initially chosen instead of a frontier-tier model since the formalisation task should not require much inventiveness/reasoning: the actual required steps are already present in the `wordplay`, the task is *merely* to translate to Python. To determine whether the choice of `Gemini-Flash` was a limiting factor, we subsequently tested an unmodified `Gemma2-9B-it` model on the same task.

[2]This rejection of invalid words here is not 'cheating' since we do not use the dictionary to suggest words, rather it is only used to weed out actively proposed non-words from a short-list.

```
Action=Enum('Action',
            'ANAGRAM,REMOVE_FIRST,INITIALS,REMOVE_LAST,'+
            'GOES_INSIDE,GOES_OUTSIDE,REVERSE,SUBSTRING,HOMOPHONE')
# External definitions
def is_synonym(phrase:str, test_synonym:str, pattern:str='') -> bool:
  # True if 'test_synonym' is a reasonable synonym for 'phrase',
  # with letters optionally matching 'pattern'
def is_abbreviation(phrase:str, test_abbreviation:str) -> bool:
  # Determines whether 'test_abbreviation' is
  # a valid abbreviation or short form for 'phrase'
def action_type(phrase:str, action:Action) -> bool:
  # Determines whether 'phrase' might signify the given 'action'
def is_anagram(letters:str, word:str) -> bool:
  # True if 'word' can be formed from 'letters' (i.e. an anagram)
def is_homophone(phrase:str, test_homophone:str) -> bool:
  # Determines whether 'test_homophone' sounds like 'phrase'
```

Figure 4: External functions available via In-Context Learning

In terms of the DSL itself, the back-end to the `is_synonym` and `is_homophone` functions consists of calls to simple language models. The `action_type` function performs a nearest-neighbour match against list of indicator words, and the `is_abbreviation` function performs a look-up against a list of abbreviations - both sourced from Deits (2022). For string manipulation actions (such as 'REVERSE'), the LLM formaliser itself was capable of producing correct string manipulation expressions unaided.

3.4 In-Context Learning

To produce Python code that could be sent to the prover, the LLM was prompted in an In-Context Learning ("ICL") manner. This consisted of the following parts:

1. Cryptic crossword rubric to explain to the LLM what the principles were behind the fields such as `clue, definition, wordplay`, etc.
2. Many-shot `clue` $\rightarrow$ `wordplay` examples (20 given)
3. The 'external functions' rubric shown in Figure 4
4. Few-shot `wordplay` $\rightarrow$ `Python` formalisations (6 examples given)
5. The actual `clue`, `answer`, `definition` and `wordplay` being formalised

Gemini-Flash did not appear to be particularly sensitive to the prompting style used, except in the 'handover step' (between problem description and model generation) where several trials were needed to obtain the final function definition in the required format consistently. Further details of all the ICL prompts are given in Appendix A.4. For the Gemma2-9B-it formalisation runs, the same prompts were used unchanged (with no other tuning/training). In additional, a further Gemma2-9B model was trained on 448 *valid* Gemini-created proofs of ground-truth Wordplay examples.

3.5 Proof Verification with Hinting

The verifier implemented here must decide whether a given formalisation is valid, and report any errors found to iteratively improve the Python code as feedback to the LLM formaliser in a cycle, as seen in Self-Debug (Chen et al., 2023), and AlphaCodium (Ridnik et al., 2024). Examples of assertion failures, with constructive hinting, are shown in Figure 5.

This cycle is repeated until a formalisation is validated (zero assertion failures, considered a 'SUCCESS' with the `answer` having been proved), or `max_rewrites=2` is reached. If no Python formalisation can be validated, then the fallback `answer` is used (defined as being the most frequent `answer` amongst the original candidates produced in the first stage of solving the `clue`).

```
AssertionError: assert: is_abbreviation('an Artist', 'RA') : \
   'an Artist' does not have a valid abbreviation; \
   'RA' is an abbreviation for : \
     artist, artillery, Royal Artillery, gunners, painter
AssertionError: assert action_type('goes crazy', Action.ANAGRAM) : \
  'goes crazy' does not suggest Action.ANAGRAM, but 'crazy' does
AssertionError: assert action_type('worked', Action.HOMOPHONE) : \
  'worked' does not suggest Action.HOMOPHONE, but may be Action.ANAGRAM
```

Figure 5: Illustrative `AssertionError` responses (with hinting) from the verifier

4 EXPERIMENTS

4.1 GEMMA2 9B `answer` CANDIDATE GENERATION

During the initial experimental phases of fine-tuning local models for the `answer` generation task it was discovered that `-base` models scored more highly than `-it` models. This might be explained by observing that instruction fine-tuning may (to some extent) penalise off-the-wall answers, which may be essential for our task. In addition, we also observed that while the Top-1 candidate from a model generating with a temperature $t = 0.5$ had high accuracy, it was beneficial to run candidate generation with $t = 1.0$ (even though the Top-1 accuracy was lower in this case) - since having a wider spread of `answer` candidates was useful for our pipeline overall.

4.2 GEMMA2 9B `wordplay` CANDIDATE GENERATION

Since `wordplay` is so flexible, it is difficult to evaluate it for accuracy against other examples (without, say, a large LLM to evaluate the differences). However, good `wordplay` should result in good formalisations, so evaluation is available on an end-to-end basis.

One key assumption in the system proposed here is that a correct `answer` should lead to interpretable `wordplay`, whereas an incorrect `answer` candidate should give rise to unformalisable/unverifiable `wordplay`. The following typical example illustrates how the correct `answer` leads readily to correct `wordplay` (the workings of this clue are illustrated in Figure 2d), whereas trials with an incorrect `answer` candidate (which was, in fact, the most frequent candidate for this `clue`) give clearly unverifiable `wordplay`:

```
clue: "wader woman has on (5)"
    definition: "{wader} woman has on" # Same for all

  answer: "HERON"  # correct answer
    wordplay: "woman (HER) has on (ON)"

  answer: EGRET    # incorrect answer - 3 trials shown
    wordplay: "woman (HER) on top of (REG - another word for on, \
               as in 'do you have the heating on?')"
    wordplay: "EG (woman has) + RET (on)"
    wordplay: "woman (HER) has on/around (EG) - a wader bird"
```

4.3 CRYPTONITE RESULTS (TOP-1 EXACT MATCH)

In this work, we focus our testing on using the Cryptonite dataset of Efrat et al. (2021) as a benchmark, with the Top-1 exact-match results shown in Table 1. As in Saha et al. (2024), due to computational constraints, we performed sampling of the validation and test sets, using fewer than the full 26k examples available. The standard deviation of these figures is $\approx \pm 1.5\%$ at 1000 samples, and $\approx \pm 3.3\%$ at 200. To determine whether the systems presented here 'beat' GPT-4o, we performed a Bayesian Item Response Theory test (Fox, 2010) to determine the probability that our results outperformed the GPT-4o (over the same samples).

Table 1: Cryptonite results : Standard splits, Top-1 answer accuracy rate

		Validation			Test		
Model	samples	Overall	Quick	Hard	Overall	Quick	Hard
Rule-based (*)	26k	8.3%			8.6%	13.5%	5.8%
T5-large (770M) FT (*)	26k	7.4%			7.6%	12.8%	3.4%
Gemma2-9B-it 5-shot	1000	5.7%	11.5%	5.2%	4.5%	10.5%	4.0%
Gemini-Flash 5-shot	1000	6.6%	12.5%	6.1%	6.5%	11.8%	6.1%
GPT-4o 5-shot	1000	**29.8%**	**45.0%**	**28.5%**	27.6%	47.4%	26.0%
Gemma2-9B FT	1000	21.7%	28.8%	21.1%	15.9%	38.2%	14.1%
Gemma2-9B freq (#=20)	1000	26.6%	31.3%	26.2%	25.5%	**55.3%**	23.1%
(AB) logprob answer	500	23.9%	35.9%	22.9%	22.7%	55.3%	20.1%
(AB) logprob wordplay	200	21.0%	15.4%	21.4%	20.5%	46.7%	18.4%
Gemini-Flash Formaliser	200	28.0%	23.1%	28.3%	**32.5%**	46.7%	**31.4%**
Gemma2 9B-it Formaliser	200	26.0%	23.1%	26.2%	29.0%	46.7%	27.6%
Gemma2 9B-FT Formaliser	200	27.0%	23.1%	27.3%	29.5%	53.3%	27.6%

Rows (*) are as reported in Efrat et al. (2021); The Hard columns are for the non-Quick clues

The 5-Shot results in Table 1 show that:

- GPT-4o (2024-11-25) gives stronger results than those of GPT-4-Turbo (2024-04-09) given in Saha et al. (2024) - so this is an updated baseline;
- The updated GPT-4o results show surprisingly strong performance on the validation split (unfortunately, the composition of this commercial model's training data is unknown);
- Gemini-1.5-Flash-001 (which was used in development of the formaliser) is not particularly good at solving cryptic clues in itself;
- The Gemma2-9B model gets a large uplift from fine-tuning on the Cryptonite training set (compare the 5-Shot figures to the later Gemma2-9B FT ones).

The `Gemma2-9B FT` accuracy figures are for the first result returned by the fine-tuned Gemma2 model. In contrast, the `Gemma2-9B freq` accuracy figures are for the most common (i.e. highest frequency) result among the Gemma2 `answer` candidates (for which 20 samples were generated for each `clue`). These voting-based results would have exceeded prior state-of-the-art results for open-licensed models on their own.

Going beyond single models, the `Gemini-Flash Formaliser` demonstrates Top-1 exact-match performance of 32.5% for the Cryptonite Test set, establishing a new state-of-the art result against the updated baselines (the Bayesian IRT results are that Gemini-Flash has a probability of 92% of being actually better than GPT-4o).

Moreover, the results of the non-fine-tuned `Gemma2-9B-it Formaliser` also (marginally) beat the previous state-of-the-art results - which is perhaps an even stronger statement about the capabilities the system described here, since in this case Gemma2-9B models have been used throughout the solving process, showing that it is be possible to achieve very competitive cryptic crossword solving results through reasoning with open-licensed models. The Bayesian IRT results are that the Gemma-9B FT model has a probability of 81% of being actually better than GPT-4o on Hard clues, 57% overall.

The formaliser results are (surprisingly) relatively worse for Quick clues. This seems to be related to the fact that the agreement/frequency-based `Gemma2 freq` model is very strong on these clues, and any 'contribution' from the formalising/verification procedure is likely to overrule a good baseline result, due to erroneous verification of 'proofs' that are not valid.

4.4 ABLATIONS

The lines in Table 1 marked '(AB)' are ablations. Both utilise the measurement of average *logprob* of the output tokens given by the relevant model.

The first ('logprob answer') shows the results of using the candidate answer generation Gemma2-9B FT model from above, with the candidate `answer` being chosen from the list of 20 possibilities according to highest *logprob*. Since answers are typically very short, this method is similar to the frequency-based selection model.

The second ('logprob wordplay') shows the results of evaluating the Gemma2-9B FT model that generates `wordplay` hypotheses, and choosing an `answer` based on the highest *logprob* according that generating model. Somewhat unexpectedly, this was not as effective as might be assumed from the generated `wordplay` seen in Section 4.2 - where the `wordplay` for wrong answers looks absurd. Examining samples of the `wordplay` most favoured by pure *logprob* order, it seems that the generating LLM finds simply-worded but *completely fictitious* `wordplay` quite likely.

Both of these ablations demonstrate that the formalisation and verification steps are essential components in our system - they cannot be shortcut by a 'dumb ranker' in the pipeline.

4.5 KNOWN LIMITATIONS OF THE SYSTEM

While the verifier implemented for this work is effective, it does not completely cover the following possible potential 'shortcuts' in the Python functions it analyses:

- The entire Python function might consist of comments : Nothing triggers `assert` - this has been partly countered by requiring the python code to include at least 2 `assert` statements
- The Python function contains conditional execution, routing around `assert` statements
- Occasionally, the hint `assert XYZ failed` results in the re-write : `assert XYZ==False`
- The proof may be logically disconnected, with left-hand-side terms not being supported / justified by right-hand-side terms in other lines of the code

These issues do not appear insurmountable, given time and effort. It should be noted that since the formalising LLM is only being used In-Context there is little chance that the above issues are being systematically abused (which would almost certainly happen if there was learning-in-the-loop in a Reinforcement Learning setting).

5 CONCLUSIONS

Clearly, the choice of cryptic crossword clue solving as the platform for reasoning experiments is unconventional. However, we have demonstrated clear success with a *system* that include both LLMs, verifiers and coding aids.

We believe that our results validate our overall approach to codification of the cryptic crossword problem domain : Generating `answer` candidates and `wordplay` suggestions followed by production of code via an LLM-based formalisation process; verification using Python code analysis tools; and iterative prompting of the LLM formaliser proved quite effective.

We were happy to discover that our development work using the Gemini-Flash LLM as a formaliser was directly transferable to a the open-licensed `Gemma2-it` model for the same role, with little loss of performance, enabling the whole pipeline to be run locally. The weakest link in the chain was, predictably, getting the 'Aha' of `wordplay` creation to work - humans can still generate wordplay that is beyond the capabilities of current models.

The authors sincerely hope that this work sparks interest in the cryptic crossword domain, which presents an array of interesting and challenging reasoning problems on which fruitful research can take place, even for those with limited computation budgets.

6 ETHICS STATEMENT

6.1 SOCIETAL IMPACT

There are many current cryptic crossword enthusiasts that would potentially not welcome AI-enabled solvers to 'take over' their favourite pastime. In particular, when taken further, this line of work would potentially disruptive to public leaderboards that rank people according to the time taken to solve puzzles 100% correctly. However, there is currently little risk of LLM cryptic solvers as being anything more than comic relief for current experts.

6.2 POTENTIAL BIAS IN FAVOUR OF NATIVE ENGLISH SPEAKERS

While the English language has a high capacity for ambiguity and wordplay overall, making this type of crossword possible, cryptic crosswords also exist in other languages (Wikipedia contributors, 2024). In addition, although solving cryptic crossword answers may be very difficult (even for native English speakers), understanding the `answer` from given `wordplay` is much simpler.

7 REPRODUCIBILITY

7.1 DATASETS AND CODE

The following outline the efforts that have been made to ensure reproducibility:

- **Datasets** - Resources such as dictionaries used, and the Cryptonite and Wordplay datasets are available online, via the sources referenced in the main text.
- **System Code & LLM Prompts** - Python code for the complete end-to-end system will be made available on publication. The prompting required for the LLM formalisation (arguably a form of programming) are included in the Appendix.
- **Models used / training** - The models referenced in this work are available with open weights (the Gemma2 models), or via API (the Gemini-Flash model, with a pinned version number). The training procedures are outlined in the text, and therefore have some degree of reproducibility. The fine-tuned Gemma2 models will be made available in any case, on publication.

7.2 COMPUTATIONAL REQUIREMENTS

The Gemini LLM was accessed by API, and the total spend to create the results in this paper was less than $100 USD, with each prompt round-trip taking around 5 seconds. The Fine-Tuning of the Gemma2 9B model took around 24 hours for a full Cryptonite training run, and 8 hours for the Wordplay dataset runs. Thus, the single-GPU model runs totalled less than $50 USD.

REFERENCES

Martin Andrews. Wordplay Dataset, 2024. URL https://github.com/mdda/cryptic-wordplay.

J Ross Beresford. The UK Advanced Cryptics Dictionary. Technical report, published online, 2000. https://cfajohnson.com/wordfinder/.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug, 2023. URL https://arxiv.org/abs/2304.05128.

Robin Deits. rdeits/cryptics github code repo. https://github.com/rdeits/cryptics, 2015.

Robin Deits. rdeits/crypticcrosswords.jl github code repo. https://github.com/rdeits/CrypticCrosswords.jl, 2022.

Avia Efrat, Uri Shaham, Dan Kilman, and Omer Levy. Cryptonite: A cryptic crossword benchmark for extreme ambiguity in language. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 4186–4192, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.344. URL https://aclanthology.org/2021.emnlp-main.344.

Jean-Paul Fox. *Bayesian Item Response Modeling*. Statistics for Social and Behavioral Sciences. Springer New York, 2010. ISBN 978-1-4419-0741-7. doi: 10.1007/978-1-4419-0742-4.

Kathryn J. Friedlander and Philip A. Fine. The grounded expertise components approach in the novel area of cryptic crossword solving. *Frontiers in Psychology*, 7, 2016. ISSN 1664-1078. doi: 10.3389/fpsyg.2016.00567. URL https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2016.00567.

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

Gemma Team and Google DeepMind. Gemma 2: Improving open language models at a practical size, 2024. URL https://arxiv.org/abs/2408.00118.

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.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timoth'ee Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, Sketch, and Prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations*, 2023. URL https://doi.org/10.48550/arXiv.2210.12283.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, December 2022. ISSN 1095-9203. doi: 10.1126/science.abq1158. URL http://dx.doi.org/10.1126/science.abq1158.

Ansong Ni, Srini Iyer, Dragomir Radev, Ves Stoyanov, Wen tau Yih, Sida I. Wang, and Xi Victoria Lin. Lever: Learning to verify language-to-code generation with execution, 2023.

Tal Ridnik, Dedy Kredo, and Itamar Friedman. Code generation with AlphaCodium: From prompt engineering to flow engineering, 2024.

Abdelrahman "Boda" Sadallah, Daria Kotova, and Ekaterina Kochmar. Are LLMs good cryptic crossword solvers?, 2024. URL https://arxiv.org/abs/2403.12094.

Soumadeep Saha, Sutanoya Chakraborty, Saptarshi Saha, and Utpal Garain. Language models are crossword solvers, 2024. URL https://arxiv.org/abs/2406.09043.

Trieu Trinh, Yuhuai Wu, Quoc Le, He He, and Thang Luong. Solving Olympiad geometry without human demonstrations. *Nature*, 2024. doi: 10.1038/s41586-023-06747-5.

unsloth.ai. Unsloth code repo. https://github.com/unslothai/unsloth, 2024.

Eric Wallace, Nicholas Tomlin, Albert Xu, Kevin Yang, Eshaan Pathak, Matthew Ginsberg, and Dan Klein. Automated crossword solving, 2022.

David Webb. Epic crossword battle expert vs. Times cryptic puzzle #29029. https://youtu.be/N5p4TqdjsHs, 2024.

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. URL https://arxiv.org/abs/2201.11903.

Wikipedia. Cryptic crossword — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Cryptic_crossword&oldid=1228427465, 2024. [Online; accessed 1-July-2024].

Wikipedia contributors. Cryptic crossword - regional variation. https://en.wikipedia.org/wiki/Cryptic_crossword#Regional_variation, 2024.

PW Williams and D Woodhead. Computer assisted analysis of cryptic crosswords. *The Computer Journal*, 22(1):67–70, 1979.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models, 2023.

A APPENDIX

A.1 CRYPTIC CROSSWORD BACKGROUND

The following borrows extensively from the description on Wikipedia (2024) (kudos to the authors there), to which we have added `wordplay` annotations in a notation typical of the `FifteenSquare.com` website (and in the Wordplay dataset use in this work).

A.1.1 BASICS

A cryptic clue leads to its answer only if it is read in the right way. What the clue appears to say when read normally (the surface reading) is usually a distraction with nothing to do with the solution. The challenge is to find the way of reading the clue that leads to the solution.

A typical clue consists of two parts:

- The straight or definition. This is in essence the same as any non-cryptic crossword clue: a synonym for the answer. It usually exactly matches the part of speech, tense, and number of the answer, and usually appears at the start or end of a clue. For our annotations, the span that encompasses the `definition` is highlighted using curly braces.
- The cryptic, subsidiary indication or wordplay. This gives the solver some instructions on how to get to the answer in another (less literal) way. The wordplay parts of clues can be obscure, especially to a newcomer, but they tend to utilise standard rules and conventions which become more familiar with practice.

Sometimes the two parts of the clue are joined with a link word or phrase such as 'from', 'gives' or 'could be'. One of the tasks of the solver is to find the boundary between the definition and the wordplay, and insert a mental pause there when reading the clue cryptically.

We list below several of the important styles of `wordplay` that are commonly used, each with an annotated example. For a more comprehensive list, along with an outline of the 'Ximenean principles', please see Wikipedia (2024).

A.1.2 ANAGRAMS

An anagram is a rearrangement of a certain section of the clue to form the answer. This is usually indicated by a codeword which indicates change, movement, breakage or something otherwise amiss. For example:

```
clue:       Chaperone shredded corset (6)
definition: {Chaperone} shredded corset
answer:     ESCORT
wordplay:   (corset)* (*shredded)
```

A.1.3 CHARADE

In a charade, the answer is formed by joining individually clued words to make a larger word (namely, the answer). For example:

```
clue:       Outlaw leader managing money (7)
definition: Outlaw leader {managing money}
answer:     BANKING
wordplay:   BAN (outlaw) + KING (leader)
```

A.1.4 CONTAINERS

A container or insertion clue puts one set of letters inside another. For example (also starting to add a little more indirection):

```
clue:       Utter nothing when there's wickedness about (5)
definition: {utter} nothing when there's wickedness about
```

```
answer:     VOICE
wordplay:   O (nothing) with VICE (wickedness) around it (about)
```

A.1.5 DELETIONS

Deletion is a wordplay mechanism which removes some letters of a word to create a shorter word. For example:

```
clue:       Bird is cowardly, about to fly away (5)
definition: {Bird} is cowardly, about to fly away
answer:     RAVEN
wordplay:   [c]RAVEN (cowardly) - 'C' (i.e. circa, about) (-fly away)
```

A.1.6 DOUBLE DEFINITION

A clue may, rather than having a definition part and a wordplay part, have two definition parts. For example:

```
clue:       Not seeing window covering (5)
definition: {Not seeing} {window covering}
answer:     BLIND
wordplay:   Double Definition (DD)
```

A.1.7 HIDDEN WORDS

With hidden word clues, the solution itself is written within the clue – either as part of a longer word or across more than one word. For example:

```
clue:       Found ermine, deer hides damaged (10)
definition: Found ermine, deer hides {damaged}
answer:     UNDERMINED
wordplay:   [fo]UND ERMINE D[eer] (hides)
```

A.1.8 HOMOPHONES

Homophones are words that sound the same but have different meanings, such as ‘night’ and ‘knight’. Homophone clues always have an indicator word or phrase that has to do with being spoken or heard. For example:

```
clue:       We hear twins shave (4)
definition: We hear twins {shave}
answer:     PARE
wordplay:   "pair" (twins, "we hear")
```

A.1.9 REVERSALS

A word that gets turned around to make another is a reversal. For example:

```
clue:       Returned beer fit for a king (5)
definition: Returned beer {fit for a king}
answer:     REGAL
wordplay:   (LAGER)< (beer, <returned)
```

A.2 WORDPLAY DATASET

The Wordplay Dataset used in this work is extracted from websites where cryptic crossword enthusiasts post solutions to the puzzles published in major publications. Each completed puzzle is annotated by an solver who provides the community with `definition`, `wordplay` and `answer` fields for each of the approximately 30 clues in that day's grid.

For UK papers, these enthusiast websites include:

- `timesforthetimes.co.uk` - Times, Times Quick
- `www.fifteensquared.net` - Independent, Guardian, Financial Times
- `bigdave44.com` - Telegraph, Sunday Telegraph

The following is an example from the Wordplay dataset, formatted in YAML (the workings of this clue are illustrated in Figure 2c):

```
title: Financial Times 16,479 by FALCON
url: https://www.fifteensquared.net/2020/05/18/ \
     financial-times-16479-by-falcon/
author: teacow
clues:
- clue: '{Offer} of support also broadcast'
  pattern: '8'
  ad: D
  answer: PROPOSAL
  wordplay: PROP (support) + (ALSO)* (*broadcast)
- ...
```

In the above:

- `clue` is the original clue, as given to solvers, but with the 'regular crossword' `definition` portion highlighted with curly braces;
- `pattern` is the number of characters in the answer;
- `ad` (across/down) is potentially significant, because some clues include directional hints such as 'before' or 'upwards' which are only meaningful if the orientation of the answer within the grid is known;
- `answer` is the clue's final answer (not known to the solvers before solving); and
- `wordplay` is an informally annotated explanation of how the clue words act together to logically build the letters in the answer (the resulting grid letters typically being in upper case) - here the `*` symbol signifies that `ALSO` is to be anagrammed due to the anagram indicator (`broadcast`) in the clue.

The Wordplay dataset is publicly available as Andrews (2024). Note that care was taken to ensure that the training/validation/test splits follow those of the Cryptonite dataset (and the test set answers are deliberately scrubbed from the retrieved data by the provided scripts, to reduce the chance that they become training data for an over-eager crawling system).

A.3 FINE-TUNING PROMPT

The following is a verbatim training example used for the fine-tuning of the `Gemma2-9B-base` model:

```
### Instruction:
Cryptic clue wordplay generation : Given the clue and the answer, \
return expert definition and wordplay annotations

### Input:
clue: "musical and ballet, oddly, that can be avoided"
answer: EVITABLE ~ evitable

### Response:
definition: musical and ballet, oddly, {that can be avoided}
wordplay: EVITA (musical) + B[a]L[l]E[t] (ballet, odd letters)
```

A.4 IN-CONTEXT LEARNING PROMPTS FOR THE GEMINI LLM

The Gemini LLM is prompted in-context with the concatenation of the following sections:

- Cryptic Crossword overview
- Many-shot wordplay examples
- Declaration of 'external' Python functions
- 6-shot formalisation demonstration
- Actual problem statement (for continuation as a Python proof)
- *After a verification failure*: Error messages for the generated proof, with hints if available, and request to improve iteratively

The sections of the prompt are described more fully below, note that care was taken to ensure that the chosen terminology was use consistently throughout.

A.4.1 CRYPTIC CROSSWORD PREAMBLE

The following is the rubric and `wordplay` preamble given to the Gemini LLM:

```
A Cryptic crossword question involves using the words in \
the given clue to yield an answer that matches the letter pattern.
The clue will provide a definition of the answer, as well \
as some 'wordplay' that can also be used to confirm the answer.
Expert question solvers write informal 'proofs' using a \
particular format.

For the definition, the original clue is annotated with \
'{}' to denote where the definition is to be found.
For the wordplay, the following conventions are loosely used:
* The answer is assembled from the letters in CAPS
* Words in brackets show the origins of letters in CAPS, \
often being synonyms, or short forms
* Action words are annotated as illustrated:
  + (ETO N)* (*mad = anagram-signifier) = TONE
  + (FO OR)< (<back = reversal-signifier) = ROOF
  + [re]USE (missing = removal-signifier) = USE
* DD is a shorthand for 'Double Definition'
```

A.4.2 MANY-SHOT WORDPLAY EXAMPLES

Around 20 examples from the Wordplay dataset are included in the in-context prompt:

```
For example:
---
clue: "arrived with an artist, to get optical device (6)"
definition: arrived with an artist, to get {optical device}
answer: CAMERA
wordplay: CAME (arrived) + RA (artist, short form)
---
clue: ...
```

A.4.3 EXTERNAL PYTHON DSL FUNCTIONS

Domain Specific Python functions are described in-context to the LLM, which appears able to use them without seeing their internal functionality. In fact, the actual implementation of the functions is more extensive than described, since calls to these functions also track 'near misses' which can be fed back as hints during the re-write process.

```
The task is to produce a formal proof using python code, \
where the docstring will also include an informal proof as an aid.
The following are functions that can be used in your output code:

Action=Enum('Action', 'ANAGRAM,REMOVE_FIRST,INITIALS,REMOVE_LAST,'+
                      'GOES_INSIDE,GOES_OUTSIDE,REVERSE,SUBSTRING,HOMOPHONE')
# External definitions
def is_synonym(phrase:str, test_synonym:str, pattern:str='') -> bool:
  # Determines whether 'test_synonym' is a reasonable synonym for 'phrase',
  # with letters optionally matching 'pattern'
def is_abbreviation(phrase:str, test_abbreviation:str) -> bool:
  # Determines whether 'test_abbreviation' is
  # a valid abbreviation or short form for 'phrase'
def action_type(phrase:str, action:Action) -> bool:
  # Determines whether 'phrase' might signify the given 'action'
def is_anagram(letters:str, word:str) -> bool:
  # Determines whether 'word' can be formed from 'letters' (i.e. an anagram)
def is_homophone(phrase:str, test_homophone:str) -> bool:
  # Determines whether 'test_homophone' sounds like 'phrase'
```

A.4.4 FEW-SHOT FORMALISATION EXAMPLES

The following are 3 (out of 6) of the few-shot formalisation examples given before the final test-case prompt:

```
The following are examples of simple functions that prove that \
each puzzle solution is correct:

```python
def proof(answer="ONCE",
          clue="head decapitated long ago", pattern='4'):
  """
  definition: head decapitated {long ago}
  wordplay: [b]ONCE (head decapitated = remove first letter of BONCE)
  """
  assert is_synonym("head", "BONCE")
  assert action_type("decapitated", Action.REMOVE_FIRST) \
         and "BONCE"[1:]=="ONCE"
  assert is_synonym("long ago", "ONCE", pattern='4')
proof()
```

```python
def proof(answer="DECIMAL",
```
```

```
          clue="the point of medical treatment", pattern='7'):
  """
  definition: {the point} of medical treatment
  wordplay: (MEDICAL)* (*treatment = anagram)
  """
  assert is_synonym("the point", "DECIMAL", pattern='7')
  assert action_type("treatment", Action.ANAGRAM)
  assert is_anagram("MEDICAL", "DECIMAL")
proof()
```

```python
def proof(answer="SUPERMARKET",
          clue="fat bags for every brand that's a big seller",
          pattern='11'):
  """
  definition: fat bags for every brand that's {a big seller}
  wordplay: SUET (fat) (bags = goes outside) of \
            (PER (for every) + MARK (brand))
  """
  assert is_synomym("fat", "SUET")
  assert action_type("bags", Action.IS_OUTSIDE)
  assert "SUET" == "SU" + "ET"
  assert is_abbreviation("for every", "PER")
  assert is_synomym("brand", "MARK")
  assert "SU"+"PER"+"MARK"+"ET" == "SUPERMARKET"
  assert is_synonym("a big seller", "SUPERMARKET", pattern='11')
proof()
```

A.4.5 Formalisation instruction

The following instruction is given before the final 'test-case' prompt illustrated in Figure 3:

```
# Please complete the following in a similar manner, and return the whole function:

```python
def proof(answer= ...
```

A.4.6 Proof Verification with Hinting

Examples of assertion failures, with constructive hinting, are shown:

```
AssertionError: assert: is_abbreviation('an Artist', 'RA') :
   'an Artist' does not have a valid abbreviation;
   'RA' is an abbreviation for : artist, artillery, Royal Artillery,
   gunners, painter
AssertionError: assert action_type('goes crazy', Action.ANAGRAM) :
  'goes crazy' itself does not suggest Action.ANAGRAM, but 'crazy' does
AssertionError: assert action_type('worked', Action.HOMOPHONE) :
  'worked' does not suggest Action.HOMOPHONE, but maybe Action.ANAGRAM

# Please re-implement the SOLUTION above \
(altering both the docstring and the python code as required), \
taking care to fix each of the problems identified, \
and return the whole function:

```python
def proof(answer= ...
```
```

Once the prover has fully parsed a given output with zero assertion failures, the proof is considered a success (up to 2 re-write iterations are allowed, more that that is considered an overall failure to prove the answer).
```