Proving that Cryptic Crossword Clue Answers are Correct

Martin Andrews [1] **Sam Witteveen** [1]

Abstract

Cryptic crossword clues are challenging cognitive tasks, for which new test sets are released on a daily basis by multiple international newspapers. Each cryptic clue contains both the definition of the answer to be placed in the crossword grid (in common with regular crosswords), and 'wordplay' that *proves* that the answer is correct (i.e. a human solver can be confident that an answer is correct without needing crossing words to confirm it). Using an existing cryptic wordplay proving framework (operating on Python proofs created by an LLM), we show that it is possible to distinguish between correct answers and almost-correct ones based upon whether the wordplay 'works'.

1. Introduction

Recent advances in computational models have significantly improved their ability to handle diverse natural language tasks involving complex syntactic and semantic interpretations. Despite these strides, machines continue to fall short of human performance in areas requiring flexible problem-solving, swift adaptation to new tasks, and effective generalization across unfamiliar domains.

This gap is particularly evident in the domain of cryptic crossword solving - a popular activity across the world, with multiple papers in the UK, Australia, India and elsewhere featuring daily puzzles for readers to solve.

The domain of cryptic crossword solving has received little attention, despite being a notable language-oriented cognitive task, with solvers worldwide. One possible reason is that cryptic crosswords are much less common in the United States than 'regular crosswords'. Another possibility is that cryptic crosswords combine a challenging cross-discipline mix of advanced language processing capabilities, logical reasoning, and an 'Aha! moment'.

[1]Red Dragon AI, Singapore. Correspondence to: Martin Andrews <martin@reddragon.ai>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

```
# (1) Statement of original problem
def proof(
      clue="arrived with an artist, \
              to get optical device",
      pattern="6",
      answer="CAMERA"):  # Provided
  """
  # (2) Hypothesised by local LM
  definition: arrived with an artist, \
                to get {optical device}
  wordplay: CAME (arrived) + \
    RA (artist, from RA = Royal Academy)
  """
  # (3) Continuation generated by LLM
  assert is_synonym("arrived", "CAME")
  assert is_abbreviation("artist", "RA")
  assert "CAME" + "RA" == "CAMERA"
  assert is_synonym(
     "optical device", "CAMERA",
     pattern="6")
proof()   # Triggers proof verification
```

Figure 1. From problem statement to LLM formalisation

The following illustrates the elements of a cryptic crossword clue (for more background please refer to the Appendix A):

```
clue:       Research done, primarily,
            on most of magical beings (5)
definition: {Research} done, primarily,
            on most of magical beings
wordplay:   D[one] (primarily) (most of)
            ELVE[s] (magical beings)
answer:     DELVE
```

In this example, the `clue` is the text given to solvers (with the number of letters in the answer in brackets). The reasoning steps include: (i) identifying the `definition` (highlighted with curly braces), similar to a regular crossword; (ii) parsing the remainder of the clue to identify the key elements of the `wordplay`. Here, for instance, there are action words like 'primarily' (meaning : take the first letter), and 'most of' (meaning : remove some letters from) that are applied to other parts of the clue; (iii) finally assembling the first letter of 'done' and most of the letters in 'elves' (the magical beings) to 'prove' that the correct `answer` is 'DELVE' (agreeing with the `definition` span).

In this work, taking a cue from the effectiveness of verifiers for other reasoning problems (Lightman et al., 2023; Jiang

et al., 2023), we approach the cryptic crossword clue solving problem as one that combines Language Models to tackle (i) the NLP elements; (ii) the creation an informal proof (i.e. coming up with `wordplay`); (iii) the formalisation process (which re-writes the `wordplay` logic in Python); and (iv) a 'prover' that can check whether the claims are justifiable. Rather than simply returning `valid/invalid`, the prover provides 'LLM-friendly' messages about validity, allowing the LLM to re-write its previous attempt iteratively.

1.1. Contributions

The following is the main contribution of this work:

- **Show the effectiveness of the proving mechanism** - By using both the true `answer` and a nearby `candidate`, we show that the prover can distinguish between them based on the provability of the `wordplay`

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 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 aid in breaking the combinatorial explosion of possibilities to achieve solving rates in the high 90% range (Wallace et al., 2022).

2.2. Cryptic Crosswords

In contrast to a regular crossword clue, a cryptic clue leads to its answer only if it is read in the right way. The clue itself contains both a conventional 'straight definition', and wordplay that can be used to derive the same answer. Once a given clue is understood, a solver can enter it into the grid with near 100% certainty, even on a standalone basis.

To get a flavour of the mental processes involved in solving these puzzles, it is highly recommended to watch an expert going through the full process for a recent Times Cryptic Crossword (including the reasoning steps in each clue) [1].

Despite cryptic crosswords being being relatively unknown in the U.S.A, globally there are active communities of solvers, with multiple daily leaderboards and annual international competitions.

[1] Cracking the Cryptic (17-May-2024) https://youtu.be/vudt7LlUX00?t=124

2.3. Cryptonite Dataset

The (UK) Times Cryptic Crossword is widely considered the gold standard in puzzles, even though they are not necessarily the most difficult, because the clues are unusually well constructed. Cryptonite (Efrat et al., 2021) is a large-scale dataset of Cryptic Crossword clues from the Times, containing 523,000 naturally sourced clues from an extended time-period, with the train, validation and testing splits chosen so that a given answer only appears in one of the splits.

2.4. Rule-based solvers

Williams & Woodhead (1979) is an early example of attempting to devise a formal language for describing cryptic clues. However, they found that the clues' linguistic elements tend to thwart such formal approaches.

Deits (2015; 2022) used a more flexible rule-based solver with a manually-crafted probabilistic grammar. Building on the assumption that a clue can usually be split into a wordplay and a definition, the (brute-force) solver tries to find the most probable parse such that the wordplay yields a semantically-similar result to the definition. Reported in Efrat et al. (2021), the rule-based solver approach yields an accuracy of 8.6% on the Cryptonite test set.

2.5. LLM-based solvers

Cryptic crossword clues seemed like an idea target for BERT-era models. However, Efrat et al. (2021) reported that a T5-Large model fine-tuned on Cryptonite's 470k cryptic clue training set achieved only 7.6% test set accuracy on the test set (i.e. below that of rule-based solvers).

Interestingly, present day (scaled) Large Language Models also score very poorly on cryptic clues. This is 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 that each clue provides.

2.6. Code & reasoning

To compensate for LLMs only approximating the 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 are generated and filtered in order to generate a valid solution, to AlphaCodium (Ridnik et al., 2024), in which solutions are iterated upon and involving much less computation, this work uses a

prover that can feed back 'hints' to the formalising LLM, so that the task of re-writing nearly-valid proofs is made easier.

3. Methods

3.1. Wordplay dataset

There are a number of websites where cryptic crossword enthusiasts post completed puzzles, annotated with `definition`, `wordplay` and `answer` fields. In order to capture these key elements of cryptic crossword clue solving, we make use of a Wordplay dataset gathered from such sites (further details in Appendix B).

3.2. Language Model set-up

In our experiments, we make use of two Language Models.

In order to generate the `definition` and `wordplay` fields, we make use of the Llama-3-it 8B model (AI@Meta, 2024), fine-tuned using LoRA (Hu et al., 2021) to generate `definition` and `wordplay` annotations from the original `clue` and (importantly) a candidate `answer`. Training on 5371 examples (with the prompt format as shown in Appendix C) took under 3 hours on a single GPU virtual machine, using the `unsloth` package (unsloth.ai, 2024).

To create the python 'proofs' of the correctness of solutions, we use both Google's `Gemini-Pro-1.0-002` and `Gemini-Flash-1.5-001` LLMs (pinned model versions to enable a level of reproducibility).

While the Llama model was found to be capable of reasonable guesses at correct `definition` and `wordplay` annotations, the creation (and iterative fixing) of the Python proofs required the use of more capable models.

3.3. Hypothesis testing

The hypothesis tested in this work is whether it is possible for the combination of Llama `definition` and `wordplay` generation; Gemini LLM formalisation; and a Python-based prover to have sufficient 'power' to distinguish between candidate answers (one of which is the correct answer). Ideally, the *correct* answer will lead to perfect `wordplay`, which then can be translated into elegant Python code, while an *incorrect* candidate answer will lead to 'bizarre' `wordplay`, which in turn will be formalised into Python that will be incapable of being proved.

3.4. Obtaining a close candidate answer

For a given question, we use the Llama model to create a `definition` and `wordplay` pair from the `clue` and the ground-truth `answer`. We then use the span in the generated `definition` to create an alternative candidate `answer` that both matches the `pattern` and is semantically close to the phrase marked in the `definition`. This closest match is obtained by filtering a list of crossword words (Beresford, 2000) sorted by cosine-similarity to the `definition` span, when both are embedded using FastText (Mikolov et al., 2018).

```
def proof(answer="RUDE",
  clue="rudeness about son's computer language",
  pattern='4'):
  """
  definition:
    {rudeness} about son's computer language
  wordplay:
     RUD[e] (about, S (son)) +
     ASS (assistant)
  """
  assert is_synonym("rudeness",
    "RUDE", pattern='4')
  assert is_abbreviation("son", "S")
  assert is_synonym(
    "assistant", "ASS") # Fails
  assert "RUD" + "ASS" == "RUDE" # Fails
proof()
# NB: correct answer is "LISP"
#   wordplay:
#     (LIP) (rudeness) about (S) (son)
```

Figure 2. Incorrect `answer` leading to formalisation failure

3.5. Formalising and proving an answer

From a candidate `answer`, we use the Llama model to generate a `definition` and `wordplay` pair. We then use the Gemini LLM to attempt to generate Python proofs, which are then verified using a Cryptic Crossword DSL expressed via Python (see Appendix D for further details). This process includes 're-writes' where the proof verifier can return errors in response to assertion failures, along with hints about how these errors might be fixed. After the initial draft proof, the verifier allows up to 5 re-write attempts to be made - until the proof is either accepted or the verification process stops (i.e. no success after 5 re-writes).

In the case of the close candidate answer, the `wordplay` is likely to be rather nonsensical - the hypothesis being tested here is whether the formalisation process can reject close candidate answers, in favour of the ground-truth answer. Figure 2 gives an illustration of the kind of output produced when a non-ground-truth `answer` is converted to `wordplay` (and then an attempt at proving it is made).

4. Experiments

4.1. Distinguishing ground-truth answers from close candidates

For each of 100 different clue examples from the Wordplay dateset, we use the ground-truth answer to generate 1 close

Table 1. Frequency of 'provability' wins by aggregation method LLM version: 1.0P is Gemini-Pro-1.0, 1.5F is Gemini-Flash-1.5

METHOD	LLM VER.	TRUE POS	DRAW	FALSE NEG
COMPLETED PROOFS	1.0P	38%	59%	3%
FASTEST SOLVE	1.0P	38%	56%	6%
MEAN SOLVE TIME	1.0P	38%	56%	6%
COMPLETED PROOFS	1.5F	40%	55%	5%
FASTEST SOLVE	1.5F	40%	55%	5%
MEAN SOLVE TIME	1.5F	42%	53%	5%

candidate answer, as in Section 3.4.

We then provide the ground-truth and the candidate answers to Llama to generate 5 different `definition` and `wordplay` samples for each.

Given the `definition` and `wordplay`, we use the Gemini LLM to formalised the problem into Python (an example of which is shown in Figure 1), and then attempt verification of that Python proof, with a maximum of 5 re-writes (attempts at re-formalisation) for each potential proof (as in Section 3.5).

Finally, we gather the results (number of re-writes required for successful proof, or a fail) across all 100 questions $\times$ 5 samples $\times$ 2 candidate answers.

To see whether the ground-truth answer was more 'provable' than the close candidate, we check which of them obtained: (a) the higher number of completed proofs (of any number of re-writes); (b) the fastest proof (i.e a proof requiring fewest re-writes); (c) the faster average solve time, where unsolved counts for 6 re-writes (rather than infinity).

5. Results

The results of testing the 'provability' hypothesis are shown in Table 1, where we show percentages of True Positive (ground-truth answer more provable), False Negative (non-ground-truth answer more provable) and Draw (both answers proved to equal extents) across the different provability measures, for each of the two Gemini models.

Clearly, the results suggest that the proving system has a degree of *preference* towards correct answers, but is a long way from being a reliable oracle of answer correctness.

This points to an issue that would likely occur if the system were scaled up to testing many candidate answers, rather than just 2 possibilities here. Specifically, if the cryptic crossword clue task were transformed to choosing between a large number of potential candidates the current system would likely start to become less accurate overall, since the number of False Negative results would likely start to dominate the True Positive results. That being said, there are many avenues for improvement, in particular solving some of the limitations outlined in the Section 6.

Looking across the LLM versions, it is also encouraging to see that the (much cheaper) Gemini-Flash model is slightly more capable of proving the ground-truth answers.

6. Limitations

The Prover does not detect a number of potential errors / problems:

- Cryptic crossword setting 'rules' dictate that the clues should contain exactly enough to prove an answer, the prover does not check that all valuable words in the clue have been utilised
- Proofs may be logically disconnected, with left-hand-side terms not necessarily being connected to right-hand-side terms in other lines of the code.
- Entire Python function consists of comments : Nothing triggers `assert`
- Python function contains conditional execution, routing around assert statements : Nothing triggers `assert`
- Occasionally, the hint `assert XYZ failed` results in a re-write `assert XYZ==False`, which is cheating

With additional effort, the authors believe that these issues are surmountable. However, since the Gemini LLM is only being used In-Context, there currently is little chance that the above issues are being systematically abused (which would almost certainly happen if there were learning-in-the-loop in a Reinforcement Learning setting).

7. Conclusions

It is increasingly hypothesised that the next-token-prediction task may be insufficient to get machines to reason and plan (Kambhampati, 2024). By framing the cognitive task of cryptic crossword solving as a reasoning problem that is addressable by LLMs supported by a verification system, this work has sought to bring this reasoning task within the scope of what is tractable by *systems* that have components that include LLMs as well as verifiers and coding aids.

The authors sincerely hope that this work sparks an interest in the cryptic crossword domain, since it presents a challenging NLP/reasoning task, with huge scope for testing different reasoning approaches. Notably, the current State-of-the-Art solving methods score less than 20% on a real-world test set.

Impact Statement

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 be 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.

Naturally, the authors also believe that the techniques here have wider applications to the field of Machine Learning, but they do not in themselves present any particular additional societal risk.

Bias towards English-language speakers

The English language has a high capacity for ambiguity and wordplay overall, making cryptic crosswords much more feasible. However, they do exist in other languages - please see the Cryptic Crossword Wikipedia page for a broader view of their worldwide prevalence. Note that deriving the answers is very difficult (even for native English speakers), whereas understanding the answer from given wordplay is much simpler.

Acknowledgements

Support for this research was provided by the Google AI/ML Developer Programs team, including access to the Gemini models and GPUs on Google Cloud Platform.

The authors thank the ICML workshop reviewers for their time and valuable feedback.

References

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

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

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

Deits, R. rdeits/crypticcrosswords.jl. https://github.com/rdeits/CrypticCrosswords.jl, 2022.

Efrat, A., Shaham, U., Kilman, D., and Levy, O. Cryptonite: A cryptic crossword benchmark for extreme ambiguity in language. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t. (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.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 10764–10799, 2023.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-Rank Adaptation of Large Language Models, 2021.

Jiang, A. Q., Welleck, S., Zhou, J. P., Li, W., Liu, J., Jamnik, M., Lacroix, T., Wu, Y., and Lample, G. 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.

Kambhampati, S. Can large language models reason and plan? *Annals of the New York Academy of Sciences*, 1534(1):15–18, March 2024. ISSN 1749-6632. doi: 10.1111/nyas.15125. URL http://dx.doi.org/10.1111/nyas.15125.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., Hubert, T., Choy, P., de Masson d'Autume, C., Babuschkin, I., Chen, X., Huang, P.-S., Welbl, J., Gowal, S., Cherepanov, A., Molloy, J., Mankowitz, D. J., Sutherland Robson, E., Kohli, P., de Freitas, N., Kavukcuoglu, K., and Vinyals, O. 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.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step, 2023.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

Ridnik, T., Kredo, D., and Friedman, I. Code generation with AlphaCodium: From prompt engineering to flow engineering, 2024.

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

Wallace, E., Tomlin, N., Xu, A., Yang, K., Pathak, E., Ginsberg, M., and Klein, D. Automated crossword solving, 2022.

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].

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

A. 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. 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.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.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.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.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.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.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.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.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)
```

B. 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:

```
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.

Code that generates the Wordplay dataset is available at https://github.com/mdda/cryptic-wordplay. Note that care has been taken to ensure that the training/validation/test splits follow those of the Cryptonite dataset (and the test set is deliberately not provided, to reduce the chance that it becomes training data itself).

C. Fine-tuning prompt

The following is a verbatim training example used for the fine-tuning of the Llama-3-it model:

```
<|start_header_id|>system<|end_header_id|>

Cryptic clue wordplay generation : Given the clue and the answer, \
return expert definition and wordplay annotations<|eot_id|>\
<|start_header_id|>user<|end_header_id|>

clue: "musical and ballet, oddly, that can be avoided"
answer: EVITABLE ~ evitable<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>

definition: musical and ballet, oddly, {that can be avoided}
wordplay: EVITA (musical) + B[a]L[l]E[t] (ballet, odd letters)<|eot_id|>\
<|end_of_text|>
```

D. 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.

D.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'
```

D.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: ...
```

D.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'
```

D.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()
```

D.5. Formalisation instruction

The following instruction is given before the final 'test-case' prompt illustrated in Figure 1:

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

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

D.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 5 re-write iterations are allowed, more that that is considered an overall failure to prove the answer).
```