Decrypting Cryptic Crosswords: Semantically Complex Wordplay Puzzles as a Target for NLP

Joshua Rozner
Stanford University
rozner@stanford.edu

Christopher Potts
Stanford University
cgpotts@stanford.edu

Kyle Mahowald
University of Texas at Austin
mahowald@utexas.edu

Abstract

Cryptic crosswords, the dominant crossword variety in the UK, are a promising target for advancing NLP systems that seek to process semantically complex, highly compositional language. Cryptic clues read like fluent natural language but are adversarially composed of two parts: a definition and a wordplay cipher requiring character-level manipulations. Expert humans use creative intelligence to solve cryptics, flexibly combining linguistic, world, and domain knowledge. In this paper, we make two main contributions. First, we present a dataset of cryptic clues as a challenging new benchmark for NLP systems that seek to process compositional language in more creative, human-like ways. After showing that three non-neural approaches and T5, a state-of-the-art neural language model, do not achieve good performance, we make our second main contribution: a novel curriculum approach, in which the model is first fine-tuned on related tasks such as unscrambling words. We also introduce a challenging data split, examine the meta-linguistic capabilities of subword-tokenized models, and investigate model systematicity by perturbing the wordplay part of clues, showing that T5 exhibits behavior partially consistent with human solving strategies. Although our curricular approach considerably improves on the T5 baseline, our best-performing model still fails to generalize to the extent that humans can. Thus, cryptic crosswords remain an unsolved challenge for NLP systems and a potential source of future innovation.

1 Introduction

Modern computational models have made great progress at handling a variety of natural language tasks that require interpreting rich syntactic and semantic structures [6; 30; 25; 32]. However, in NLP [2; 26; 3] as in other areas of AI [22], machines still lag humans on tasks that require flexible problem solving, rapid learning of unseen tasks, and generalization to new domains. Just as complex games mastered by human experts, such as chess, Go, and video games, have proved a fertile ground for developing more flexible AI [36; 35; 27], we propose that creative language games are a rich area for developing more flexible NLP models. In particular, we argue that linguistic tasks involving meta-linguistic reasoning pose an important and significant challenge for state-of-the-art computational language systems.

One such domain is cryptic crossword puzzles. Cryptics are the most popular style of crossword in the United Kingdom and appear in major newspapers like *The Times* and *The Guardian*. Cryptic clues have two parts: a **definition** and a **wordplay** cipher that, when placed adjacent to each other, read like fluent natural language. For example, consider this NLP-centric cryptic crossword clue: "But everything's really trivial, initially, for a transformer model (4)" with answer "BERT". "(4)" is the **enumeration** and specifies the number of letters in the answer. Solvers must identify which

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

Table 1: Examples of five common clue types in cryptic crosswords, all demonstrating clues for the answer: BERT. Indicators, where they occur, are italicized. The wordplay substrate appears in bold. Typographical emphasis added to aid reader, only; actual clues have no such indication.

Clue type	Clue example	Explanation for this example
Anagram: An anagram clue requires scrambling some set of clue letters to produce the answer.	*Confused*, **Bret** makes a language model (4)	*Confused* indicates that we should "confuse" (anagram) the letters of "Bret" to get BERT.
Initialism: An initialism requires taking the first letters of a phrase	**B**ut **e**verything's **r**eally **t**rivial, *initially*, for a language model (4)	*initially* indicates taking the first letters of "But everything's really trivial".
Hidden: The answer occurs within a larger phrase.	Language model *in* som**ber t**ext (4)	*in* indicates that a word is hidden inside a phrase. Extract the word BERT from the phrase "somber text."
Charade: For a charade clue, each part of the answer is clued sequentially.	A language model exist? Right time! (4)	"exist" becomes "BE" since "exist" and "be" are synonyms. A standard abbreviation for "right" is "R." A standard crossword abbreviation for "time" is "T." This clue type does not have an indicator.
Double definition: In a double definition clue, two synonyms or phrases appear next to each other, each of which can refer to the answer.	Model Sesame Street character (4)	Bert is a valid answer for "Sesame Street character", and it is also a model. Double definitions do not have indicators.

part of the clue is definition and which is wordplay. The **definition** should be a semantically valid description of the answer word: "a transformer model" can mean "BERT". The **wordplay** part is the remainder of the clue: "But everything's really trivial, initially". The word *initially* in this context is used as an **indicator**, which tells the solver to take the *initial letter* of each of the preceding words (**b**ut **e**verything's **r**eally **t**rivial) to produce "BERT". Because both the wordplay and the definition give the same 4-letter word (which is what the **enumeration**, "(4)", calls for), we can be confident that "BERT" is the correct answer.

The clue just discussed is an example of the *initialism* clue type, which is one of roughly 15 major clue types. Other types can require anagramming words, finding words hidden in longer phrases, performing synonym substitutions, substituting a word for a soundalike (e.g., "hiring" for "high ring"), or composing a number of these manipulations. See Table 1 for examples of several other clue types with answer "BERT" and Appendix A for examples of actual clues from the dataset. As with American-style clues, cryptics require world knowledge and linguistic flexibility to match the definition, but also considerable attention to meta-linguistic concepts to solve the wordplay. In this paper we study the language task of solving individual cryptic clues, rather than full puzzles since, unlike American-style crosswords, cryptics generally have unique answers and so are less reliant on grid constraints in order to achieve a complete solution.

While cryptics pose a challenge to novice solvers unfamiliar with their structure, experts flexibly combine world, domain-specific, and linguistic knowledge to solve novel clues. Experts know the rules that govern cryptic clues, but they also reason about them flexibly and apply them to solve novel clues. In the psychology literature, it has been claimed that cryptics depend on domain-general intelligence and are an ideal domain for studying the "aha moment" in humans [11; 10] because experts can solve them with high accuracy (but rarely at first glance), they can be easily generated, and they require drawing on a diverse range of cognitive abilities. Therefore, we believe that cryptic crosswords are an excellent domain for developing computational language systems that "learn and think like humans" [22], posing an interesting and important challenge for modern machine learning.

Our main contributions are, first, a cleaned dataset of cryptic crosswords clues from theguardian.com, consisting of 142,380 clues from 5,518 puzzles over 21 years;[1] and second, a novel curriculum learning approach, in which the model is first fine-tuned on related, synthetic tasks (e.g., an augmented word descrambling task) before tackling actual cryptic clues. This method meaningfully improves on a standard T5 seq-to-seq approach and on the best model of Efrat et al. [7]—concurrent work that presents a similar dataset and similar neural baseline using T5.

In this paper, we aim not only to present a dataset and propose a novel solution but also to characterize the problem and motivate its importance. To that end, we elucidate the task with three non-neural baselines and fine-tune T5 [31], a Transformer-based [38] encoder-decoder, as a neural baseline. Since the character-level wordplay inherent to cryptics might be challenging to language models with subword tokenization (T5 uses SentencePiece [21]), we study whether T5 has or can acquire meta-linguistic knowledge. In Section 6.1 we examine whether T5 learns meta-features of the task related to answer length. In Section 6.3 we use a descrambling task to assess whether T5 understands the character composition of words and whether the model can make use of linguistic and meta-linguistic information simultaneously. Our results show, perhaps surprisingly, that the subword-tokenized T5 model is quite robust to character-level challenges. Moreover, the descrambling task may serve as a useful benchmark task in guiding the development of new approaches on the cryptics task.

Given the compositional nature of cryptic clues, we investigate the extent to which the model generalizes under increasingly difficult data splits. In Section 6.2 we introduce a new form of disjoint data split to address T5's robustness to inflection: a word-initial disjoint split that segments clue–answer pairs based on the first two letters of the answer. In Section 6.4 we examine the systematicity of the model's answer generation by perturbing the wordplay portion of anagram clues, showing that its behavior is partially consistent with human solving strategies.

Although our novel curricular approach considerably improves performance on this task, fully solving cryptics remains a challenge for modern machine learning, with expert humans still outperforming machines. Therefore, we hope that our dataset will serve as a challenging benchmark for future work.

2 Related work

While there is an existing literature on puns and wordplay in NLP [17; 20; 24] as well as on solving American-style crosswords [13; 23; 34], there has been relatively little work using NLP methods on cryptics, which require processing highly compositional language and disambiguating surface-level from hidden meaning. Hart and Davis [16] laid out a computational framework for solving the problem, and Hardcastle [14, 15] proposed some rule-based solutions. The app Crossword Genius from Unlikely AI solves cryptic clues and gives human-understandable explanations. Because its method is proprietary and not available for open testing, we do not report it as a baseline but note that it is competitive. Deits [5] offers a rule-based solution, which can also output explanations, and which we test on our dataset. Most recently, in work concurrent to ours, Efrat et al. [7] release a similar dataset of cryptic crossword clues and present a neural baseline using T5-large. In Section 6.5, we discuss how our new data split and curricular approach improve on their work.

Our curricular approach fits into the space of recent work on pre-finetuning [37; 1] and curricular approaches for compositional tasks [12; 40]. Our approach is loosely related to the pre-finetuning of Aghajanyan et al. [1] but differs in that our curriculum is composed of fewer tasks that are all closely related to the primary task and synthetically generated. Our approach resembles Geva et al. [12] in that we attempt to endow language models with a specific kind of reasoning by training in a multi-task setup over synthetic data. In the vein of Wu et al. [40], which trains on synthetic datasets to encode inductive bias, our approach can be understood as encoding wordplay functional biases.

Whether large pretrained Transformer models generally pick up on meta-linguistic features like word length and the character composition of words is an open question. Brown et al. [4] explore a set of restricted word unscrambling tasks in the few-shot (versus fine-tuned) setting for GPT-3. Probing results from Itzhak and Levy [18] suggest that information about the character composition of words is present in the embeddings of pretrained, subword-tokenized models. This seems to

[1] We release the dataset along with all code to reproduce the results in this paper at https://github.com/jsrozner/decrypt.

confirm our result that subword-tokenized models do have more knowledge of character composition and meta-properties of words than we might have expected.

3 Dataset and task

We present a cleaned dataset of 142,380 cryptic crossword clues from 5,518 puzzles published in *The Guardian* from July 1999 to October 2020. We also introduce a challenging "word-initial" disjoint split after observing that T5 is robust to inflection. Overall, the dataset has 55,783 unique answers, giving a mean frequency of 2.55 clues per unique answer. Answers in the dataset consist of one to six words, with most (97.4%) having one (83.3%) or two (14.1%) words. Full details of dataset preprocessing are given in Appendix B, and in addition to releasing the full dataset, we include code to fully replicate our data download, pre-processing pipeline, and split generation in the repository.

3.1 Task

We frame the problem as a standard seq-to-seq task from inputs (clues with length enumeration) to outputs (answers). For example, one input could be *But everything's really trivial, initially, for a transformer model (4)*, with output *BERT*. This is consistent with how clues are presented to human solvers. Full details of the input–output specification for each model are provided in Appendix C, along with other experimental details.

3.2 Splits

As motivated in the introduction (and in Section 6.2), we consider three splits. The **naive (random) split** is a standard 60/20/20 random split into train, dev, and test. The **disjoint split** ensures that all clues with the same answer appear in only one of train, dev, or test. For example, if any clue for which the answer is "BERT" is in the train set, then *all* clues for which the answer is "BERT" are in the train set. The disjoint split is used to test composition and generalization. It prevents an approach that relies only on lexical similarity across clues (like KNN) from succeeding on the task. Finally, the **word-initial disjoint split** is designed to address T5's robustness to inflection. For this split, we enforce that all clues whose answers start with the same two letters will appear in the same set. For example, all clues that have answers starting with 'ab' like "abacus," "abdicate," "abdicates," will be grouped, ensuring that inflections or spelling variations of the same base word occur in a single split.

3.3 Metrics

Our primary metric is whether the top-ranked output is correct, *after* filtering to outputs of the correct length. We filter because each clue in a puzzle has a hard length constraint, i.e., a human solver *cannot* pencil in a solution that is of the wrong length. Additionally, we report how often the correct answer is contained in the top 10 outputs after length filtering. This is a useful metric since, when clues are presented in the context of a full puzzle, solvers use information from interlocking answers to narrow down a set of candidates. For instance, the best-performing crossword solver for American-style (non-cryptic) crosswords relies heavily on satisfying grid constraints [13]. Comparing post-length-filter results from Section 4.5 with pre-filter results from Section 6.1, the length filter is seen to increase the top-10 metric by roughly 6% for T5 (with length enumeration given).

4 Baseline models and results

To characterize how simpler approaches perform on this task, we test three non-neural baselines: a WordNet-based heuristic model, a k-nearest-neighbor bag of words model (KNN BoW), and a rule-based model designed for the task [5]. For a neural baseline, we fine-tune T5-base [31]. For models that can interpret the answer-length enumeration as a textual token (KNN and T5), we append it to the input string (e.g., "`(4)`"). For these two models, we report results both with and without appending the enumeration. Implementation details are discussed in Appendix C.

4.1 WordNet

Our first baseline is a simple heuristic based on clue structure. It takes advantage of the fact that the definition part of a cryptic clue always appears at the beginning or end of the clue. For instance, in the double definition clue for "BERT" in Table 1, "Model Sesame Street character," the word "model" appears at the beginning and is a definition (in this case, a hypernym) for the answer "BERT". We use WordNet [8], a large database of English word meanings, to build a reverse dictionary via the synonym, hyponym, and hypernym graphs. We take as candidate solutions the set of reverse dictionary outputs for the first and last words of a clue. For example, if "dog" appears at the start of a clue, candidates would include "animal", "canine", "labrador", etc. Ranking outputs by character-level overlap with the rest of the clue slightly improves performance, since answers sometimes are rearrangements of the characters in the wordplay portion of the clue.

4.2 KNN BoW

To assess whether clues that are close in lexical space have similar answers, we use a KNN model on top of a bag-of-words featurizer.

4.3 Rule-based

Finally, to evaluate how well a rule-based approach, with a hand-coded grammar, can perform on the task, we use the Deits [5] solver.[2] This solver handles anagrams, initialisms, substrings, insertions, and reversals in a rule-based way. While the rule-based version includes common clue types, an inherent limitation of this approach is that it is difficult to enumerate all possibilities. For instance, the Deits solver does not include charade-type clues, nor double definitions. Moreover, the rule-based solver uses WordNet's [8] word similarity functionality to rank outputs, meaning that, in general, it will fail on clues that have definitional parts consisting of more than one word (e.g. "`language model`" from our example in the introduction would not be matched).

4.4 T5: vanilla seq2seq

For our baseline neural seq-to-seq approach, we fine-tune the Transformer-based [38] T5-base model [31], starting from HuggingFace's [39] pretrained model parameters. T5 is an encoder-decoder language model pretrained on the C4 corpus [31]. Fine-tuning is done via supervised learning (teacher-forcing) over standard seq-to-seq input-output pairs. At test time, we generate outputs using beam search. As described in Section 3.3, we filter the outputs to those of the correct length and evaluate by checking whether the top result is correct and whether the answer occurs in the top 10. See Appendix C for details, including hyperparameter selection.

4.5 Results

In Table 2, we report metrics for dev and test sets on both the naive (random) split and word-initial disjoint split (discussion on disjointness in Section 6.2). While the WordNet baseline achieves some success (2.6% and 10.7% top-1 and top-10 on the test set), it is inherently limited, since it cannot handle clues with multiword definitions and lacks a good ranking mechanism. KNN does better, achieving 6.1% and 11.3% with length enumeration. The rule-based solver achieves 7.3% and 14.7%, marginally outperforming the KNN baseline. Though our T5 neural baseline outperforms all non-neural baselines, achieving 16.3% and 33.9%, it leaves considerable room for improvement.

5 Curriculum learning

Solving cryptic clues uses different linguistic and reasoning abilities compared to the natural language tasks on which T5 is pretrained. Thus, during fine-tuning on the cryptics task, the model must learn many sub-parts of the problem simultaneously: how to look up definitions, how to identify wordplay indicators, how to perform wordplay operations, etc. Although these elements of the task

[2] Deits [5] has a more recently implemented solver in Julia that was used by Efrat et al. [7]. We use the Python version, which may have small differences from the Julia version and is reportedly much slower (see Appendix C for more details).

Table 2: Results for baselines and top curricular approach. Details on the curricular approach are given in Section 5. Metrics are percentages calculated over the top ten model outputs, after filtering to outputs of correct length.

	Naive (random) split				**Word-initial disjoint split**			
Model	**Top correct**		**Top 10 contains**		**Top correct**		**Top 10 contains**	
	dev	test	dev	test	dev	test	dev	test
WordNet	2.8	2.6	10.8	10.7	2.6	2.6	10.6	10.5
Rule-based	7.2	7.3	14.8	14.7	7.4	7.3	14.9	14.5
KNN (no lengths)	5.6	5.6	9.9	10.1	0.0	0.0	0.0	0.0
KNN (lengths)	6.0	6.1	11.2	11.3	0.0	0.0	0.0	0.0
T5 (no lengths)	15.3	15.6	29.4	30.0	0.9	1.0	4.8	5.1
T5 (lengths)	16.0	16.3	33.1	33.9	1.1	1.1	5.6	5.8
Curricular: ACW + ACW-descramble	**21.5**	**21.8**	**42.2**	**42.4**	**6.1**	**6.5**	**18.9**	**20.0**

each individually benefit from T5's natural language pretraining, our standard T5 baseline suggests that learning to compose them all at once is challenging. We show that a curriculum of synthetic datasets can address this, substantially improving performance on the primary cryptics task. We test a number of different curricular tasks and task sets and discuss which are most effective. This curricular approach may be useful in other settings where focused linguistic capabilities are required.

5.1 Curricular datasets

The logic of our curricular approach is to provide some guidance on the sub-parts of the problem space before building up to fully compositional cryptic clues. For curricular training, we create four datasets designed to improve performance and elucidate what makes a good curricular task for this problem. We process a public American crossword clue dataset [29], henceforth "ACW-data" for "American Crossword", and generate three datasets: ACW, ACW-descramble, ACW-descramble-word. After preprocessing, ACW-data has 2.5m clue–answer pairs with 250k unique answers, giving a mean frequency of roughly ten clues per unique answer. Unlike cryptic clues, American crossword clues often involve relatively straightforward synonym or definition substitions, so the ACW dataset can be used to train definition lookup. We also produce a separate anagram dataset from a publicly available English dictionary. Details to produce all datasets are included in Appendix D.1. In all example clues that follow, the target output is "PETAL" and we use labels (prepending a word and colon) to help the model distinguish tasks.

1. **ACW**: ACW-data dataset in input–output form (i.e., a seq-to-seq version of American-style crossword clues) with no augmentation. For example, *phrase: flower part (5)*.

2. **ACW-descramble**: For each clue–answer pair in ACW-data, we create an input that models a cryptic clue by scrambling the answer word and prepending or appending it to the clue portion. For example, we scramble "petal" and randomly place it at the beginning (*descramble: etalp flower part (5)*) or end (*descramble: flower part etalp (5)*) of a clue.

3. **ACW-descramble-word**: A seq-to-seq task that is just the descrambling of answer words. When compared to ACW-descramble, this elucidates the importance of curricular and primary task similarity, in particular whether teaching a bare descrambling task is helpful to the model. Example: *descramble word: etalp (5)*.

4. **Anagrams**: Using a dictionary as starting point, we synthesize an anagram dataset: we pair a word (to be anagrammed) with an anagram indicator (e.g., "mixed up", "drunk", "confusingly") and ask the model to produce a valid anagram (i.e., a scrambled version of the word that is itself also a valid word). For example, *anagram: confusingly plate (5)* (rearrange the letters of 'plate' to get "PETAL"). The anagram dataset simulates the anagram type of wordplay in cryptic clues, with definition part omitted.

Table 3: Curricular results (left) and sample metrics for meta-linguistic feature analysis (right)

(a) Curricular approaches on the naive (random) split. Metric is correctness of top-output (5 beams with length filter).

Curricular dataset	Percent correct full dev set	Percent correct anagram subset
Baseline (no curricular)	15.7	13.7
ACW	18.3	14.4
ACW-descramble	13.1	21.4
ACW + ACW-descramble	**20.2**	24.0
ACW + ACW-descramble-word	17.8	18.3
ACW + anagram	17.1	19.1
ACW + ACW-descramble + anagram	20.1	**27.1**

(b) Sample metrics calculated over top 10 outputs *without* length filter, using naive split.

Model	% sample contains answer (top-10, no filter)		% outputs with correct length		% outputs correct word count	
	dev	test	dev	test	dev	test
KNN						
– (no lengths)	6.5	6.6	13.4	13.3	70.7	70.7
– (lengths)	10.6	10.7	85.4	85.3	89.7	89.6
T5-base						
– (no lengths)	19.0	18.8	16.0	16.2	74.2	74.1
– (lengths)	27.5	28.1	48.3	48.5	97.9	97.9

5.2 Methods

Our curricular approach is as follows: using the same methods as in Section 4.4, first, we fine-tune T5 on one or more supplementary tasks. Second, we continue fine-tuning on the primary cryptics task, periodically showing batches from the supplementary task(s), to decrease the likelihood of catastrophic forgetting [9]. Full details of our curricular training setup are in Appendix D.2.

5.3 Results

In Table 3a, we report results for our curricular approach. Overall, we find that curricular training using ACW + ACW-descramble is best and report in Table 2 a primary task improvement from 16.3% to 21.8% on the random split and from 1.1% to 6.5% on the word-initial disjoint split.

We begin by testing a curriculum with only ACW, which corresponds roughly to teaching the definitional lookup. Observing that ACW improves performance over the baseline, we test ACW-descramble, which is ACW augmented with a scrambled word component (an implicit wordplay). Surprisingly ACW-descramble leads to a decline in performance relative to both ACW and to Baseline. On the other hand, combining ACW + ACW-descramble leads to our best performing approach, demonstrating that a particular curricular combination can improve over two separate tasks. We also compare ACW + ACW-descramble to ACW + ACW-descramble-word. The drop in performance suggests that inputs that are more similar to the final task are more useful than, e.g., a bare descrambling task. To isolate the effect of task choice, all curricular approaches use the same data distribution, train for the same number of curricular fine-tuning steps, and are reviewed during primary training at the same frequency.

Finally, in order to explore whether the curricular training improves performance across the board or just on the clue types directly involved (e.g., anagrams), we report performance on an algorithmically-labeled anagram subset ("anagram subset" column in Table 3a). Adding the Anagrams subtask to the curriculum improves anagram performance but interestingly does not improve performance compared to the top curriculum, ACW + ACW-descramble.[3] We see a similar gain in anagram performance (but drop in overall performance) when training with only ACW-descramble. This suggests that pretraining for a particular clue type can improve performance on that type, but perhaps at the expense of performance on other clue types.

Beyond providing guidance to T5 on the problem sub-parts, this approach also partially addresses the disjointness of train and test sets. For the word-initial disjoint split, by periodically refreshing the curricular task, we remind T5 to produce outputs that are not in the training set of the cryptic split.

[3] The distribution and size of the Anagrams dataset is different from the ACW datasets, so we separate curricula involving the Anagrams dataset in Table 3a.

6 Model analysis

6.1 Learning meta-linguistic properties: output length and number of words

This task requires not just linguistic but also *meta-linguistic* reasoning. We investigate how model behavior changes when the enumeration (the specification of the length of the answer) is appended to the end of the input string as a number in parentheses. Whether large pretrained transformer models generally pick up on meta-linguistic features like word length is an open question.

To study whether the models learn length information, we report how often the top-10 candidate outputs for each clue are the correct length and have the correct number of words *before* applying any length filter. For both the KNN and the T5 models, we find that including the length enumeration improves overall performance, as can be seen in Table 2. (We omit the WordNet and rule-based approaches from this discussion since they have no capacity to learn the meaning of the length enumeration.)

In columns 3 and 4 of Table 3b we see that both KNN and T5 pick up on length information, generating more outputs of the correct length when the enumeration is provided. T5 is particularly proficient at producing the correct number of words in outputs. (Recall that multiword answers are indicated with an enumeration that has two numbers separated by a comma, as in "(4, 6)", indicating an answer like "ALAN TURING".) Given that T5 produces 97.9% of outputs with the correct number of words, it seems plausible that the model is learning a correspondence between the enumeration and the presence of a space or spaces in its output.

6.2 Disjointness

Based on the performance of the KNN model on the naive data split, we see that some clues can be solved by picking up on lexical similarity to other clues. Thus, we investigate whether T5 is also picking up on similarity to previously seen clues or if it is learning something about the compositional and functional structure of the cryptic clues.

To assess how much a Transformer-based model like T5 relies on having seen similar clues for the same word, we segment performance on the random split by whether the answer was in the train set. In Table 4a, we see that performance drops from 16% on the full dev set to only 3.0% on the clue subset not seen during training, confirming our intuition that lexical similarity between clues with the same answer plays a role in model success.

To formalize this result, we create and train on the two disjoint datasets described in Section 3: the basic disjoint and the word-initial disjoint splits. The T5 model achieves 3.3% accuracy on the basic disjoint split (dev) and only 1.1% accuracy on the word-initial disjoint split. The drop is likely partially attributable to robustness to inflection, since inflected forms often start with the same two letters.

6.3 Wordplay: minimal task

We investigate the extent to which T5, which uses SentencePiece tokenization [21], can perform wordplay-esque tasks like descrambling. We run an experiment as follows: we start with the ACW dataset from Section 5, further restrict to outputs with targets in a dictionary (i.e., no multiword answers), and downsample to 180k clues (10%). We create two descrambling tasks. The first is a direct descramble task, where the input is a scrambled version of the target word (e.g., *etalp* for target *petal*). The second task is a descramble with phrase tasks, in which we append the clue of the clue-answer pair after the scrambled answer letters (e.g., input is *etalp* | *flower part* for target *petal*). The second task is designed to mimic the cryptic setup, where we have a wordplay (in this case, the implicit descramble function) whose output is conditioned on a (possibly phrasal) synonym. See Appendix E.1 for more details.

We present results in Table 4b. We observe that the model reasonably learns the task on a random split (63.8%) but fails on a word-initial disjoint split (3.7%). Notably, including a phrasal definition alongside the scrambled letters in the input improves outcomes, suggesting that the model is simultaneously incorporating both meta-linguistic character-level and overall word-embedding information. This task can serve as a benchmark for models to solve the cryptic task, since it roughly upper-bounds how well a model can solve wordplays.

Table 4: Disjointness results (left) and descrambling results (right)

(a) T5-base performance (% for top-10 outputs after length filter) on naive, subset of naive not seen in train, disjoint, and word-initial disjoint splits.

Dataset	Top correct		Top 10 contains	
	dev	test	dev	test
Naive (random) split				
– Entire split	16.0	16.3	33.1	33.9
– Subset not in train	3.0	2.8	9.5	9.7
Disjoint splits:				
– Naive disjoint	3.3	3.2	12.6	12.9
– Word-initial disjoint	1.1	1.1	5.6	5.8

(b) Descrambling task, with and without phrasal definition. Metric is % for top ten outputs without length filter.

Split and task	Top correct	Top 10 contains
Random split		
– Descramble	63.8	91.6
– Descramble w/ phrase	79.4	91.2
Word-initial disjoint split		
– Descramble	3.7	12.5
– Descramble w/ phrase	12.4	24.4

Given that we observe a considerable drop from the random to disjoint data splits, we test whether T5 can learn the identity function under a disjoint split. We find that the model achieves 100% accuracy on a direct copy task in which the model is trained to simply output the string that is given as input. This suggests that T5's meager performance for descrambling on the disjoint split is not due to an inability to generate an output that has not been seen in fine-tuning.

6.4 Assessing systematicity

We investigate the extent to which the model's behavior is consistent with the compositional and systematic nature of human solving strategies. Consider our anagram clue from Table 1: "Confused, Bret makes a language model (4)," for which the answer is "BERT". Using a publicly available list of 18,000 first names, we algorithmically identify clues in the dev set that require the solver to compute an anagram of a first name, and we use those clues to compose two specialized test sets. In the first set, we scramble the original letters (e.g., *Bret* becomes *Treb*). If the model is behaving optimally, performance on this set should not suffer: what is important about *Bret* in this clue is not its semantics but its characters. In the second set we substitute another name of the same length (e.g., *John* for *Bret*). We pick first names because swapping a name does not change the grammaticality or linguistic validity of clues. Here, for a human solver, we would expect a major hit in performance since the correct answer is a valid anagram of *Bret* but not of *John*, and so the clue is no longer a valid cryptic clue. It does, however, still have a valid definition part ("a language model"), and so, if the model is picking up only on the definition part and ignoring the wordplay part, it might still do well. See Appendix E.2 for more task details.

On this set of clues, prior to any modification, we find a baseline accuracy of 33.3%. When scrambling the name, accuracy drops moderately, to 25.2%. When substituting a name with different letters, accuracy falls to 7.0%. This difference in performance on the substitution and scrambling tasks suggests that the model is, to some extent, correctly picking up on the need to use the letters of the name as the wordplay substrate. This is confirmed by the average character overlap between the altered word and the generated candidate answers. We observe 51.2% multiset overlap for the baseline (name unchanged from original clue), 51.0% for substitution of names with the same letters, and 31.4% when substituting names with different letters. For our example clue, this means that we would expect high character overlap between *Bret* and the candidate answers in the baseline set, but high overlap between *John* and the candidate answers in the substitution test set. These results suggest that, like human solvers, the model is sensitive to character manipulations at the location of the wordplay substrate.

6.5 Comparison to Efrat et al.

In contemporaneous work, Efrat et al. [7] present a dataset of cryptics from two other major newspapers and fine-tune T5-large for the task. While Efrat et al. [7] conclude that train/test disjointness is important, they do not fully consider T5's robustness to plural and other inflections. The word-initial disjoint split that we present addresses this. In particular, their 'naive' split is the same as our naive split, and their 'official' split is the same as our (naive) disjoint split. To demonstrate that our split is

Table 5: Performance of T5-large as reported by Efrat et al. [7], in our replication of their work, and with our top curricular approach (ACW + ACW-descramble). Metric is correctness of top output (5 beams without length filter) on test set.

Split	Efrat et al	Our replication of Efrat et al	Top curricular
Efrat 'naive' (test)	56.2	53.2	52.1
Efrat 'official' (test)	7.6	10.9	**19.9**
Word-initial disjoint (test)	–	4.9	**12.8**

relevant to the Efrat work, we replicate their results (we train T5-large and report the same metric, correctness of top output with b=5 beams), show a considerable decline in performance under the word-initial disjoint split (10.9% to 4.9%), and finally demonstrate that our curricular approach substantially improves results on the 'naive-disjoint' (10.9% to 19.9%) and word-initial disjoint splits (4.9% to 12.8%). Performance on the 'naive' split does not change considerably with our curricular approach. Results are in Table 5, and further training details are given in Appendix E.3.

7 Conclusion

In this paper we introduce a dataset of cryptic crossword clues that can be used as a challenging new benchmark task and develop a novel curricular approach that considerably improves performance on the benchmark. We further characterize the problem with three non-neural baselines and provide methods for investigating model behavior, including a simple descrambling task and an experiment that explores what T5 learns about compositional task structure. Lastly we introduce a challenging word-initial datasplit to evaluate a model's ability to achieve compositional generalization. These contributions demonstrate why this task is worth studying and how it may be relevant to related problems. For example, our curricular approach may be useful in other settings where focused linguistic capabilities are required.

Pretrained contextual embedding models like T5, which draw on a rich base of lexical and linguistic knowledge from pretraining, are a promising candidate for the type of flexibility needed to solve this sort of puzzle. However, T5 does not initially succeed at the task, and although our curricular approach considerably improves task performance, cryptics remain an unsolved problem. Although one might initially think that a character-level tokenization scheme would be necessary for this task, Sections 6.1 and 6.3 suggest that T5 *can* unscramble words (under appropriate generalization splits) and *does* learn a correspondence between a word's tokens and the word's total length.

Given the success of our curricular approach, future work might combine new synthetic datasets under a learned curriculum schedule. In any case, an approach that fully solves this problem will need to more flexibly learn different kinds of wordplay functions and how to functionally compose them to produce the answer word. In that sense, we believe that the cryptic crossword task serves as a good benchmark for those interested in building NLP systems that can apply linguistic and meta-linguistic knowledge in more creative, flexible, and human-like ways.

Acknowledgments and Disclosure of Funding

This work was supported in part by a Stanford HAI Hoffman–Yee grant.

We thank Saul Pwanson for guidance on the release of crossword-related datasets and William Tunstall-Pedoe for discussions on cryptics and for testing his proprietary solver on our dataset. We also thank Armando Solar-Lezama, Josh Tenenbaum, Osbert Bastani, and other members of the Neurosym group for helpful feedback.

References

[1] Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. Muppet: Massive multi-task representations with pre-finetuning. *arXiv preprint arXiv:2101.11038*, 2021.

[2] Emily M Bender and Alexander Koller. Climbing towards NLU: On meaning, form, and understanding in the age of data. In *Proc. of ACL*, 2020.

[3] Yonatan Bisk, Ari Holtzman, Jesse Thomason, Jacob Andreas, Yoshua Bengio, Joyce Chai, Mirella Lapata, Angeliki Lazaridou, Jonathan May, Aleksandr Nisnevich, Nicolas Pinto, and Joseph Turian. Experience grounds language, 2020.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

[5] Robin Deits. rdeits/cryptics, 2015. URL `https://github.com/rdeits/cryptics`.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] Avia Efrat, Uri Shaham, Dan Kilman, and Omer Levy. Cryptonite: A cryptic crossword benchmark for extreme ambiguity in language, 2021.

[8] C. Fellbaum. *WordNet: An electronic lexical database*. MIT Press, Cambridge, MA, 1998.

[9] Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999. ISSN 1364-6613. doi: https://doi.org/10.1016/S1364-6613(99)01294-2. URL `https://www.sciencedirect.com/science/article/pii/S1364661399012942`.

[10] Kathryn J Friedlander and Philip A Fine. The penny drops: Investigating insight through the medium of cryptic crosswords. *Frontiers in psychology*, 9:904, 2018.

[11] Kathryn J Friedlander and Philip A Fine. Fluid intelligence is key to successful cryptic crossword solving. *Journal of Expertise*, 3(2):101–132, 2020.

[12] Mor Geva, Ankit Gupta, and Jonathan Berant. Injecting numerical reasoning skills into language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 946–958, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.89. URL `https://aclanthology.org/2020.acl-main.89`.

[13] Matthew L Ginsberg. Dr. fill: Crosswords and an implemented solver for singly weighted csps. *Journal of Artificial Intelligence Research*, 42:851–886, 2011.

[14] D Hardcastle. Using the bnc to produce dialectic cryptic crossword clues. In *Corpus Linguistics 2001*, pages 256–265, 2001.

[15] David Hardcastle. *Riddle posed by computer (6): the computer generation of cryptic crossword clues.* PhD thesis, Citeseer, 2007.

[16] M Hart and Robert H Davis. Cryptic crossword clue interpreter. *Information and Software Technology*, 34(1):16–27, 1992.

[17] He He, Nanyun Peng, and Percy Liang. Pun generation with surprise. *arXiv preprint arXiv:1904.06828*, 2019.

[18] Itay Itzhak and Omer Levy. Models in a spelling bee: Language models implicitly learn the character composition of tokens. *arXiv preprint arXiv:2108.11193*, 2021.

[19] Brad Jascob. bjascob/lemminflect, 2019. URL `https://github.com/bjascob/LemmInflect`.

[20] Justine T Kao, Roger Levy, and Noah D Goodman. The funny thing about incongruity: A computational model of humor in puns. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 35, 2013.

[21] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[22] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.

[23] Michael L Littman, Greg A Keim, and Noam Shazeer. A probabilistic approach to solving crossword puzzles. *Artificial Intelligence*, 134(1-2):23–55, 2002.

[24] Fuli Luo, Shunyao Li, Pengcheng Yang, Baobao Chang, Zhifang Sui, Xu Sun, et al. Pun-gan: Generative adversarial network for pun generation. *arXiv preprint arXiv:1910.10950*, 2019.

[25] Christopher D Manning, Kevin Clark, John Hewitt, Urvashi Khandelwal, and Omer Levy. Emergent linguistic structure in artificial neural networks trained by self-supervision. *Proceedings of the National Academy of Sciences*, 2020.

[26] Gary Marcus. The next decade in AI: Four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177*, 2020.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[29] Saul Pwanson, 2021. URL `http://xd.saul.pw/data/`.

[30] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[31] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL `http://jmlr.org/papers/v21/20-074.html`.

[32] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works. *arXiv preprint arXiv:2002.12327*, 2020.

[33] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

[34] Noam M Shazeer, Michael L Littman, and Greg A Keim. Solving crossword puzzles as probabilistic constraint satisfaction. In *AAAI/IAAI*, pages 156–162, 1999.

[35] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[36] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[37] Oyvind Tafjord, Matt Gardner, Kevin Lin, and Peter Clark. QuaRTz: An open-domain dataset of qualitative relationship questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5941–5946, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1608. URL `https://aclanthology.org/D19-1608`.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf`.

[39] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

[40] Yuhuai Wu, Markus Rabe, Wenda Li, Jimmy Ba, Roger Grosse, and Christian Szegedy. Lime: Learning inductive bias for primitives of mathematical reasoning. *arXiv preprint arXiv:2101.06223*, 2021.

A Clue examples from dataset

Table 6: Examples from our dataset, taken from the train portion of the naive split. Replicates Table 1 in the main paper. Indicators, where they occur, are italicized. The wordplay substrate appears in bold. Typographical emphasis added to aid reader, only; actual clues have no such indication.

Clue type	Clue example	Explanation for this example
Anagram: An anagram clue requires scrambling some set of clue letters to produce the answer.	Honour, **Ben** and **Noel** *with a new order* (4)	*with a new order* indicates that we should re-order (anagram) the letters of "Ben" and "Noel" to get ENNOBLE.
Initialism: An initialism requires taking the first letters of a phrase	*Initially*, **i**s **d**octor **e**lated **a**t result of brain operation (4)	*initially* indicates taking the first letters of "is doctor elated at", which gives IDEA, the "result of brain operation".
Hidden: The answer occurs within a larger phrase.	Cryptic advice f**or a cl**ever *solver to extract* (6)	*solver to extract* indicates that a word is hidden inside a phrase. Extract the word ORACLE from the phrase "for a clever".
Charade: For a charade clue, each part of the answer is clued sequentially.	Nitrogen and oxygen shown to exist to student chemist (5)	"Nitrogen" becomes "N", "oxygen" becomes "O", "shown to exist" becomes "BE" since they are synonyms, and a standard abbreviation for student is "L" for learner. NOBEL was a chemist! This clue type does not have an indicator.
Double definition: In a double definition clue, two synonyms or phrases appear next to each other, each of which can refer to the answer.	Painful withdrawal, having raw meat (4,6)	"COLD TURKEY" means both "Painful withdrawal" and "raw meat". Double definitions do not have indicators.

B Cryptics dataset preprocessing

To produce the clean dataset, we remove 15,591 clues that interact with other clues in the same puzzle as follows:

1. 7,687 clues that are explicitly marked as being part of a clue grouping (i.e. clues that the puzzle author has explicitly marked as interacting). For example, from Guardian puzzle 21633:[4]

   (a) 20-across: *this cast no blight on semi-conventional party (8,8)*
   "SCOTTISH NATIONAL"

   (b) 5-down: *see 20*

   In this case, the answer must be written into two locations (20-across and-5 down). The first part (20-across) is a valid clue for our models, but we omit clues of this type because programatically parsing them would require simultaneously looking at multiple clues during preprocessing.

2. 607 "continuation" clues or clues that are part of an annotated grouping: These include clues that start with an asterisk (indicating clue grouping) or those that start with an ellipsis, which indicates continuation from a previous clue. For example, from the same puzzle:

   (a) 23-across: *drunken kilty whams a dram ... (4,6)*
   "MALT WHISKEY"

   (b) 24-across: *..and another, by the sound of it, on a 20 isle, while ... (4)*
   "RHUM"

[4] Each puzzle can be accessed at `https://www.theguardian.com/crosswords/cryptic/puzzle_id`.

Solving 24-across requires having seen 23-across. ("Rum" is a type of malt whiskey that sounds like Rhum, which is a Scottish isle.) Note that we also needed to substitute Scottish for 20, from 20-across.)

3. 7,066 clues that contain a numeral. Many clues with a numeral are references to a solution in another part of the puzzle (i.e. the other solution must be substituted for the numeral). Some numerals are not references, but distinguishing them programatically is not straightforward, so we omit them. See for example, the substitution required above of "Scottish" for "20".
4. 90 clues that do not match our regular expression or with an empty clue after regular expression extraction.
5. 56 where the answer does not match the length enumeration.
6. 85 where there are unrecognized characters in the clue (e.g., unparsed HTML).

We further remove 1,611 exact duplicates. These are clues with the same target answer and clue strings that match after lower-casing, normalizing whitespace, normalizing articles ("a", "an", "the"), and stripping off punctuation.

In addition to releasing the full dataset, code to fully replicate our data download and pre-processing pipeline is also available in the GitHub repository. The code we provide reproduces this detailed information, including removal counts broken down by reason, whenever it is run to generate the data splits; comments in the code provide additional details.

C Baseline experiment details

We provide details of model and task set-up, hyperparameter choice, machines and compute used, and evaluation methods.

Evaluation is the same for all models. When evaluating the correctness of outputs, we lowercase all letters and ignore whitespace. Generated whitespace (i.e., spaces between generated multiword answers) is considered only for evaluating the meta-properties (e.g., number of words) for model outputs. The GitHub repository includes code to exactly replicate all evaluations.

C.1 WordNet

The WordNet heuristic approach produces candidate outputs as follows: the first and last words of a clue are extracted from the clue and lowercased. For each of these two words, we do a reverse dictionary lookup using WordNet. We try building the reverse lookup with synonyms, hyponyms, and hypernyms, where the last two have controllable lookup depth (e.g., hypernyms of the first set of hypernyms, etc). Any underscores or hyphens in WordNet lookup results are replaced with spaces. We test with and without inflection of lookup results by using [19] to produce all possible inflections. We filter to outputs of the correct length, excluding whitespace. We try ranking outputs (1) by by their multiset character-level overlap with the rest of the clue (i.e. not the word used for the reverse lookup), (2) by bigram overlap with the rest of the clue using a modified Levenshtein distance, and (3) by the order in which they are added to the output set (i.e., without further ranking). For this model, the number of generated outputs is determined by changing which parts of the WordNet graph (synonyms, hyponyms, hypernyms, and depth) we use to generate candidates.

This model does not involve any training, so the train set is not used. We take the configuration that produces the best performance on the dev set: we use reverse lookup with synonyms and hyponyms to depth 1, omit inflected forms, and rank using multiset character-level overlap.

We can upper-bound this method by observing that, when including synonyms and hyponyms/hypernyms up to depth three, and inflecting all outputs using LemmInflect [19] (i.e., producing the maximum number of candidates for each clue), our definition sets contains the correct answer 22% of the time. This performance could be achieved if we had a perfect ranking function. However, since our ranking mechanism is poor, we do not achieve this level of performance and find that the best outcome is achieved by reducing the size of our reverse dictionary space to include only synonyms and hyponyms to depth 1.

The slowest of these models is the one with full hyponym/hypernym lookup to depth 3 and was run on a 2013 Macbook Air in two minutes.

C.2 KNN BoW

The KNN model is implemented with scikit-learn's [28] CountVectorizer and KNeighborsClassifier. The CountVectorizer lowercases all characters and considers only alphabetic characters, numbers, parentheses and the | character. All other characters function as split locations and are themselves omitted. When including the length enumeration we append length as, e.g., *(4)* or *(4|6)*, in the case of multiword solutions. We use '|' so that the length enumeration is treated as a single token. As for all other traininable models, targets for the train set are lowercase solutions with spaces separating multiword answers. We select the 3000 nearest neighbors for each test clue so that we always produce at least ten outputs of the correct length for each clue.

We train by fitting the train set and take the set of hyperparameters that produces the best performance on the dev set: in particular, we use 1-grams, since performance degrades with longer n-grams.

This model was run on a 2013 Macbook Air in roughly ten minutes.

C.3 Rule-based

We run the Deits [5] solver on our clue sets. The model is not trainable, so we directly evaluate it on our dev and test sets. We follow Deits' guidance to set up our clue file, providing a text file where each line is of the form, *clue* | *answer* – for example,

But everything's really trivial initially for a transformer model (4) | bert

We do not restrict the number of outputs generated by this model.

The rule-based solver uses a context free grammar (CFG) that specifies possible clue forms. For example, a grammar for an anagram clue type could be "$Anagram $AnagramIndicator $Definition". Terminals for $AnagramIndicators (and other types of indicators in the full grammar) come from custom lists of indicators. One of the components of the CFG is a definition: the definition terminal is matched to a word or set of words. The non-definition part of the grammar ("$Anagram $AnagramIndicator" in the above example) is evaluated to produce possible wordplay outcomes (in this case, computing valid anagrams of the tokens matched to the $Anagram terminal). Finally, the possible wordplay outputs are compared to the definitional tokens using WordNet's word similarity function. Parses with higher similarity are ranked higher.

As mentioned in the footnote in Section 4.3, Deits [5] has a more recently implemented solver that is reportedly faster. Because the Python solver is slow, we set a timeout of 120 seconds (timed-out runs usually still produce some candidate answers) and report an average time to solve a clue of roughly 40 seconds. This results in a total time to evaluate each of the dev and test sets of approximately 300 CPU hours. We evaluate this model using multiple internal cluster CPUs run in parallel.

C.4 T5: vanilla seq2seq

Starting from HuggingFace's [39] pretrained model parameters, we fine-tune T5-base to produce the target answer for each clue input. As described in Section 3.1, inputs are of the form, e.g., *But everything's really trivial, initially, for a transformer model (4)*, with output *bert*.

We optimize with Adafactor [33] using the relative step and warmup initialization options, as implemented in the HuggingFace library (all other parameters are left unchanged from the HuggingFace defaults). We use a batch size of 256 input–output (clue–answer) pairs with per-batch truncation-to-longest, which is implemented by HuggingFace's T5FastTokenizer. We train with a patience of 15 epochs and select the best model according to dev set performance, based on whether the top answer (over 5 beam search generations) is correct. During evaluation, we generate 100 outputs for each input (100 beams with 100 output sequences) in order to evaluate sample metrics. Hyperparameters, including those for generation (max-length=10 tokens, length-penalty=0.05), were selected based on dev set performance. This setup, including all hyperparameters, is implemented in the code that we release on GitHub.

We use an internal cluster. Training takes approximately 100 minutes on a single GeForce RTX 3090 GPU. Evaluation takes roughly 120 minutes.

D Curriculum learning

D.1 Datasets

D.1.1 ACW-data

ACW-data is the unprocessed version of the American crossword clue dataset [29]. To preprocess it, we

1. Remove clues that do not match our reverse-dictionary goal: We remove clues that contain underscores or multiple hyphens, since these are generally fill-in type clues, rather than phrasal synonyms. We remove clues that reference other clues, i.e., those containing "Across" or "Down" in the clue text. We remove clues likely to be abbreviations, i.e., those with a clue ending in a period with an answer fewer than four letters, since cryptics rarely include abbreviations. We remove clues where the clue text ends in a question mark.
2. We attempt to make the clues resemble our dataset by removing any periods that occur at the end of clues, since cryptic clues do not generally have periods at the end of normal clues (though they do admit other types of punctuation).
3. We filter normalized duplicates using the same approach as for cryptic clues (i.e. clues with the same clue and answer strings after normalizing case, whitespace, and articles and stripping punctuation.

This produces a cleaned dataset of 2,464,951 clue-answer pairs from which we produce the three ACW-data-derived datasets used in curricular training. It is worth noting that some of the answers in this dataset are multiword answers that are unsplit. Optimally we would find a way to split these answers to increase similarity to our primary dataset, which does split multiple word targets.

The code to reproduce this preprocessing and to produce the following datasets is included in the GitHub repository. Details of the three datasets (ACW, ACW-descramble, and ACW-descramble-word were given in the main paper (Section 5.1).

D.1.2 ACW training datasets

The actual input-output pairs for ACW, ACW-descramble, and ACW-descramble-word are produced from the processed version of ACW-data at train time. At train time, we prepend a task label as described in the main text. The ACW curricular dataset has no further modification. For ACW-descramble and ACW-descramble-word, we produce a scrambled version of the letters during dataset collation and modify the input as specified in the main text. The provided code includes the collation functions that produce the final input–output pairs for these three datasets.

D.1.3 Anagrams dataset

First, we produce a list of valid English words to be considered for anagramming from a publicly available dictionary of English words. Using this list of words, we group all words into whether they are anagrams of each other (i.e. grouping them by their sorted letters). For anagram indicators, we use Deits [5] list of anagram indicators.

This produces 13,535 anagram groups (i.e., 13,535 unique sets of letters from which can be realized at least two valid English words). These groups contain a total of 32,305 total words. The anagram indicator list has 1,160 anagram indicators. At train time, a curricular epoch consists of showing each anagram group to the model once. To do this, during collation at train time, we randomly sample two of the anagrams from each set, randomly sample an anagram indicator, and randomly sample a position (prepend or append).

D.2 Training

As described in Section 5.1, each supplementary dataset has its own task label [31], which is passed to the model as part of the input string, and all inputs include length enumeration as in the vanilla T5 case. We fine-tune T5-base in the same way as described in Appendix C.4, but with the following modifications.

For curricular training, we first fine-tune on one or more supplementary tasks according to a training schedule, for which we tune the following hyperparameters: the number of curricular epochs, the frequencies with which each task is shown, whether the Adafactor optimizer is reset before primary training (only affects non-constant LR, i.e., when we are training T5-base but not when we are training T5-large), and the frequency of curricular subtask review during primary training. We hand-tune these hyperparameters, generally finding that training nearly to convergence on a held-out dev set for the primary curricular task is optimal. We also find that, for T5-base, resetting the optimizer between curricular and main training slightly improves performance. The specific configurations to replicate curricular training are included in the GitHub repository.

In order to directly compare the different curricula, we set up the curricula so that the number of training examples shown to the model in each epoch as well as the mix between curricular and primary task are the same. For example, for our single-dataset curricula (ACW and ACW-descramble), we run experiments with 4 curricular epochs and relative batch frequences (primary dataset: curricular dataset) during main training of 20:6. When training on curricula that include two curricular datasets, we do only 2 curricular epochs and use relative batch frequencies of 20:3:3 (primary: curricular 1: curricular 2).

To produce Table 3a, we evaluate only on the dev set over five generations to enable faster iteration. To produce the second column of the table, we algorithmically identify anagram clues. Code to replicate the anagram labeling and evaluate on this subset is available in the GitHub repository.

To produce our top result in Table 2, we double the total number of curricular epochs (from 2 to 4), select the best model checkpoint via dev set performance, and perform final evaluation on the test set taking 100 generations.

For all curricular training we use an internal cluster. Each curricular epoch takes roughly 150 minutes, giving a total curricular training time of roughly ten hours. Primary training afterward takes roughly 130 minutes since we continue to review the curricular datasets. This gives a total train time of roughly 12 hours on a single GeForce RTX 3090 GPU.

E Model analysis details

E.1 Descrambling task

We start with the preprocessed version of ACW-data from Appendix D.1.1 and further remove any clue–answer pair with an answer that is not in an English dictionary (e.g., multiword answers would be removed). This guarantees that all descrambling targets are valid English words.

After removing multiword answers, we have a dataset of 1,796,078 clues. We keep only words that have between 4 and 14 characters and downsample to 10% (roughly 180k clue-answer pairs).

We train T5-base to complete the descrambling tasks using the same approach as in Appendix C.4. Code to replicate dataset creation, training, and evaluation are available in the GitHub repository.

E.2 Wordplay systematic learning

Detailed code that identifies first name anagram substrates and generates substitutions is included in the GitHub repository. For name identification, we use names lists from the US Naval Academy and the U.K. Office of National Statistics (both lists, including with download URLs are provided in the GitHub repository). We identify 27 clues in the dev set and 69 clues in the train set that require anagramming a single word that is also a first name, and for each we perform 10 scramble and 10 name substitutions.

E.3 Efrat et al training

We use the same training setup as in Appendix C.4, but with the following changes: we train T5-large with a constant learning rate of 3e-5 and an effective batch size of 768. For evaluation we use the same metric (top output with b=5 beams, no filter) as used by Efrat et al. [7].

We again train on an internal cluster using a single GeForce RTX 3090 GPU. Training to replicate Efrat et al. [7] results (i.e. non-curricular) takes roughly ten hours. Curricular pretraining is done for

3 epochs and takes roughly 4 hours per curricular epoch, giving a total time for curricular pretraining of roughly 12 hours.

Code to replicate this approach is included in the GitHub repository.