NEUROSYMBOLIC GENERATION OF VALID CRYPTIC CROSSWORDS WITH PROOF-GUIDED CLUE GENERATION AND CONSTRAINT SOLVER DRIVEN GRID CONSTRUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Cryptic crosswords are crossword puzzles where each clue contains both a semantic definition for the solution and a wordplay from which the solver should construct the solution, for instance, using anagrams, homophones or hidden words. As in a regular crossword, the solutions are also connected together in an interlinking grid. Generating puzzles that satisfy this complex set of linguistic and combinatorial constraints presents an interesting challenge for natural language processing. In this work, we evaluate the ability of LLMs to generate valid cryptic crosswords. We break this problem down into two tasks: clue generation and grid construction, and demonstrate both tasks are still beyond the current abilities of state-of-the-art LLMs. However, whilst LLMs alone cannot solve this problem, we show that by combining LLMs with a domain-specific proof language, a set of validation tools and a satisfiability solver, we can generate cryptic crossword puzzles that are both syntactically and semantically valid. Our results show that LLMs alone are unreliable generators of complex constrained objects, but their performance improves when they are required to express intermediate reasoning in a formal language, and when correctness is delegated to deterministic validators and constraint solvers.

1 INTRODUCTION

Large language models (LLMs) produce fluent, plausible text yet struggle to satisfy complex interacting constraints. The task of cryptic crossword generation provides an example of this. A cryptic crossword clue encodes its answer through two independent routes: a semantic *definition*, which functions like a standard crossword clue, and a *wordplay* component, which specifies a sequence of operations (anagrams, reversal, etc.) whose result matches the answer. A clue is valid only when the wordplay is well-formed and its result matches the definition, with every word of the clue contributing to one component or the other. At the puzzle level, these answers must additionally fit into an interlocking grid in which crossings agree and all entries satisfy the grid's structural constraints

For example, the clue "Angered by grenade exploding (7)" yields "ENRAGED" in two ways. Angered provides the definition, while exploding indicates that the letters of grenade should be rearranged. A valid clue therefore requires agreement between a semantic relation at the sentence level and exact transformations at the character level. Existing token-level constrained-decoding techniques—whether based on context-free grammars Scholak et al. (2021) or automata Zhang et al. (2024)—cannot capture this combination of letter-level and sentence-level constraints.

We study whether current LLMs can satisfy these constraints directly, and whether explicit symbolic structure can make generation more reliable. We divide the problem into two tasks: clue generation, where the system must produce a valid cryptic clue and its answer, and grid construction, where a collection of answers must be placed into a valid crossword grid. Our experiments show a substantial gap between fluent generation and constraint satisfaction. In direct clue generation, GPT-4o and GPT-5.2 produce valid clues in only 1% and 3% of cases respectively. GPT-5.5 performs much better, reaching 95% correctness, but produces fewer unique answers. Grid construction remains

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

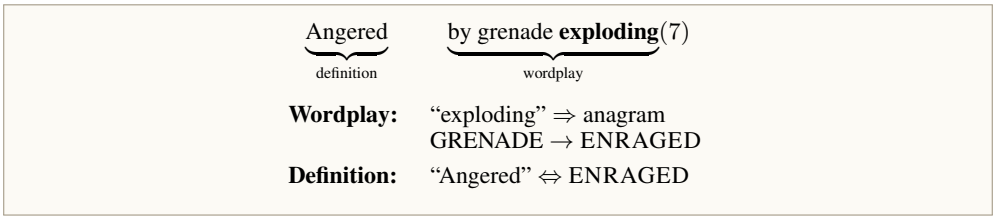


Figure 1: A cryptic crossword clue, with answer enraged.

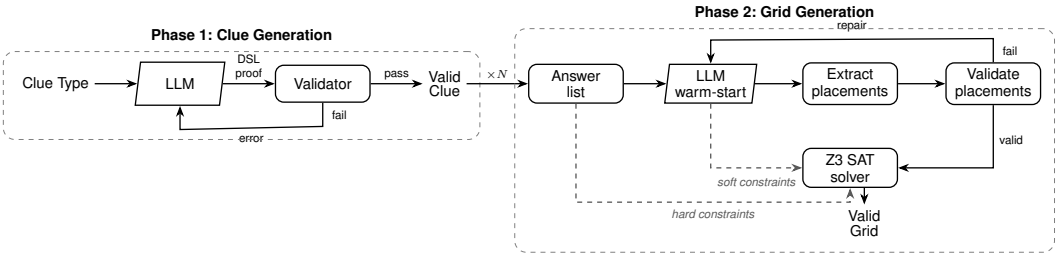


Figure 2: Full system pipeline.

difficult for all tested models: none produces a valid 13×13 crossword grid from any of the evaluated answer sets.

We address these limitations with a neurosymbolic pipeline. For clue generation, LLMs produce executable derivations in a domain-specific language that are automatically validated and repaired. For grid generation, we encode crossword validity as a satisfiability problem and use LLM-proposed placements as optional search guidance. Our system substantially improves clue validity for weaker models and generates valid grids for every answer set tested.

The main contributions of this work are: 1) An empirical evaluation of direct LLM generation for cryptic crosswords, covering both clue generation and grid construction. 2) A proof-guided clue-generation method in which LLMs generate executable DSL derivations that can be automatically checked and repaired. 3) A SAT formulation of crossword grid construction, together with two methods for incorporating LLM-proposed placements as search guidance. 4) A comparison of generated clues against gold standard human-written clues using metrics adapted from natural language processing.

2 Generating Clues

2.1 Defining a Valid Cryptic Crossword Clue

In order to tackle clue generation, we first define what constitutes a *valid* cryptic crossword clue. Regular crossword clues are primarily constrained by the semantics of the English language: a clue gives a definition or phrase that suggests the answer. For example, the standard crossword clue "Angered (7)" could admit multiple answers ("annoyed", "irate", etc.); without further information it is unclear which fits the grid. Cryptic crosswords resolve this by requiring the answer to satisfy both a definition and a wordplay simultaneously. Figure 1 illustrates this structure: only ENRAGED is recoverable from both routes, so the answer is unique. This is an example of an anagram wordplay. Several other common wordplay types are shown in Table 1.

We adopt the Ximenean criteria for validity (Macnutt, 1966). A valid clue must satisfy the following conditions: 1) it contains both a definition and a wordplay component; 2) these components are logically distinguishable, though they may partially overlap; 3) all indicators used in the wordplay accurately signal their intended operations; 4) the clue is grammatically coherent both as a surface reading and as a set of instructions; 5) no word is extraneous, with every component contributing to either the definition or the wordplay; 6) any abbreviations or synonyms employed are standard and widely accepted; and 7) the enumeration matches the length of the solution.

Clue type	Clue example	Explanation for this example
Anagram: Scrambling some set of clue letters to produce the answer.	Angered by grenade *exploding* (7)	*Exploding* suggests that we should rearrange the letters of 'grenade' to get ENRAGED, which fits the definition 'angered'.
Container: Some letters are placed within others.	Company *shelters* socialist belief (5)	A substitute for socialist is RED, and a substitute for company is CO. If we *shelter* RED inside CO we get CREDO, meaning belief.
Homophone: Some set of clue letters is a homophone of the answer.	Flower, *reportedly*, is baking ingredient (5)	*Reportedly* tells us to find a homophone of flower to get the baking ingredient FLOUR.
Deletion: Removing letters from some set of clue letters to produce the answer.	Celebrity beginning *without* end (4)	Another word for beginning is START which, if written without its end, becomes STAR (meaning celebrity).
Hidden: Answer occurs within a larger phrase.	Beer discovered in village restaurant (5)	*In* indicates that a word is hidden. Extract the word LAGER (a synonym for beer) from the phrase 'village restaurant.'
Charade: Each part of the answer is clued sequentially.	Useful fellow is odd-job worker (8)	'Useful' is a synonym of HANDY, and 'fellow' is a synonym of MAN. When put together this makes HANDYMAN (an odd-job worker). This clue type does not have an indicator.
Double definition: Two synonyms or phrases appear next to each other, each of which can refer to the answer.	Bill for attack? (6)	CHARGE is a synonym for both 'bill' and 'attack'. Double definitions do not have indicators.
Mixed Type: Any combination of the above operations can create a valid clue.	Relative of old woman and university teacher keeping son (8)	A GRAN is an old woman, a DON is a university teacher, and an abbreviation for son is S. If we charade GRAN and DON and then keep S inside, we get GRANDSON.

Table 1: Examples of common clue types in cryptic crosswords. All clues in this table were generated by our pipeline.

We refer to clues that adhere to these rules as Ximenean-valid. Any Ximenean-valid clue can be solved using only the information given in the clue and general knowledge. This is in contrast to regular crossword clues which often require one to know the answer to other intersecting clues to determine the correct solution. Note that we distinguish between valid and high quality: a clue is *valid* if it satisfies the formal Ximenean structural criteria. A valid clue is *high-quality* if it additionally has a natural surface reading, appropriate difficulty, and effective misdirection. In clue generation, we focus on validity; quality is discussed in the evaluation.

2.2 Automating Validation

To make cryptic clue validity checkable, we use a domain-specific language that represents wordplay structure explicitly, extended and adapted from prior work on cryptic crossword solving (Andrews & Witteveen, 2025a;b). The DSL is implemented as executable Python assertions. Each line asserts one component of a cryptic derivation, such as a synonym relation, an abbreviation, an indicator, or an anagram. For example, the clue in Figure 1 would produce the following proof:

```
def proof(answer="ENRAGED",
          clue="Angered by grenade exploding (7)"):
  assert action_type("exploding", Action.ANAGRAM)
  assert is_anagram("grenade","ENRAGED")
  assert is_synonym("angered","ENRAGED", pattern='7')
```

The DSL supports checks for synonyms, abbreviations, action indicators, anagrams, and other wordplay operations. Synonym checking determines whether a word or phrase can define another, including cases where one term denotes a category containing the other. Abbreviation checking determines whether a short form is a valid abbreviation of a longer phrase. Action-type checking determines

Assertion Type	Meaning	Validation Method
`is_synonym(x, y, pattern)`	Checks that `x` is a valid synonym for `y` and that `y` has the length given by `pattern`. Used for definitions and charade components.	First checks the CrosswordQA database of 6M+ clue–answer pairs (Wallace et al., 2022); falls back to an LLM query for creative or cryptic synonyms. Rejects `y` if its length does not match `pattern`.
`is_abbreviation(x, y)`	Checks that `x` is a standard abbreviation of `y`. Used for charade components.	Accepts if `x` is an initial letter of `y`; otherwise consults a curated cryptic abbreviation list (Beresford & Longair, 1992–present); falls back to an LLM check.
`action_type(x, A)`	Checks that phrase `x` signals cryptic action `A` (e.g., ANAGRAM, REVERSE, GOES_INSIDE, GOES_NEXT_TO).	Looks up `x` in an indicator dictionary (Deits, 2018); uses fastText embeddings (Joulin et al., 2017) to match unseen indicators by cosine similarity; falls back to an LLM query.
`is_anagram(x, y)`	Checks that `x` and `y` are anagrams of each other.	Normalizes both strings (uppercase, spaces removed), sorts characters, and compares. Reports missing or surplus letters on failure.
Python string ops	All other wordplay: charade concatenation, deletions, reversals, substring extraction.	Evaluated as standard Python string assertions, e.g., `"B"+"R"+"ITTEN" == "BRITTEN"` or `"aware"[::-1] == "erawa"`.

Table 2: Complete list of DSL assertion types and their corresponding validation methods.

whether a clue phrase signals a particular cryptic operation, such as `ANAGRAM`, `REVERSE`, `GOES_INSIDE`, or `GOES_NEXT_TO`. Anagram checking sorts the strings and compares their character sequences.

The validation for each assertion type can then be treated as an independent check, e.g., we can check that `"angered"` is a synonym for `ENRAGED` independently of the other assertions. This results in an executable proof of clue validity. A clue is accepted only when the corresponding proof executes without errors. The validator thus acts as a validity oracle: it does not generate the clue, but it decides whether the LLM-generated derivation satisfies required formal constraints.

A complete list of all DSL assertion types and their corresponding validation is given in Table 2. The validator checks the assertions in the proof sequentially. The solution length is enforced by passing the expected character count as a `pattern` argument to **`is_synonym`** (e.g., **`is_synonym`**`("angered", "ENRAGED", pattern='7')`), which rejects any candidate whose length does not match. A final pass checks that every word in the clue surface is accounted for by at least one assertion, preventing under-specified proofs.

2.3 The Generator Validator Loop

Once we have a validator, we can construct a generator-validator loop, as shown in Figure 2. We prompt an LLM to generate a clue of a given cryptic clue type, using the following prompt. We ask the LLM to generate the clue directly in the proof DSL, for two reasons: firstly, prior work suggests that generating in formal languages can improve even the one-shot correctness of LLMs; and secondly because it avoids the need for a potentially brittle or error prone translation step from natural language clues to the proof DSL. A prompt template is shown below.

> You are a cryptic crossword compiler and formal verifier.
>
> Your task is to generate an original cryptic crossword clue (of a [clue type] type) together with a FORMAL, STRUCTURED PROOF of the wordplay.
>
> [Description of the DSL and all available assertion types]
> [Description of clue validity requirements]
> [An example proof in the required output format]
> Task: Generate one original cryptic crossword clue (of a [clue type] type)and its complete structured proof that satisfies all of the above constraints.
> [Description of output format]

If the validation fails, we return the failure to the LLM and request that it fixes the error. We run the loop until a valid clue is returned, or a maximum number of iterations is reached.

3 GENERATING GRIDS

Given a set of clues, we now tackle the challenge of combining these words into a crossword grid, where all words are connected, and all unbroken sequences of letters in a horizontal or vertical line correspond to a valid word in the set. This problem is commonly studied in constraint-satisfaction, and is widely known to be NP-hard, and classic constraint solvers typically scale poorly as the grid size increases. We present a neurosymbolic approach to grid generation, which uses LLMs to suggest placements for the symbolic solving process, based on our hypothesis that words of certain lengths and letter patterns are more likely to be placed in certain locations in the grid, and that LLMs can learn these patterns from their training data.

3.1 SAT ENCODING OF VALID GRIDS

We define a valid grid as a satisfiability problem in which each validity rule is represented by one or more constraints.

Let $n \in \mathbb{N}$ be the grid side length and let $\Sigma = \{A, \ldots, Z\}$ denote the Latin alphabet. We introduce an empty symbol $\bot \notin \Sigma$ and write $\hat{\Sigma} = \Sigma \cup \{\bot\}$. A crossword grid is a function $G : \{1, \ldots, n\}^2 \to \hat{\Sigma}$. Let $\mathcal{W} = \{w_1, \ldots, w_k\} \subset \Sigma^+$ be a finite set of non-empty words. A placement is a tuple $\pi = (w, r, c, o)$ with $w \in \mathcal{W}$, $(r, c) \in \{1, \ldots, n\}^2$, and $o \in \{H, V\}$. Its occupied cells are $\{(r,\ c + \delta) : 0 \le \delta < |w|\}$ if $o = H$ and $(r + \delta,\ c) : 0 \le \delta < |w|\}$ if $o = V$.

We introduce a boolean placement variable $p_{w,r,c,o}$ for every possible placement of every word. $p_{w,r,c,o}$ is true iff word w is placed at (r, c) with orientation o. We introduce grid variables $g_{r,c} \in \hat{\Sigma}$, which evaluate to the character in the cell in grid row r, column c.

The entire word is placed. If a word is placed, each occupied cell must match the corresponding letter. $\forall w, r, c\ \ p_{w,r,c,H} \ \Rightarrow\ \bigwedge_{n=0}^{|w|-1} g_{r,\,c+n} = w[n] \wedge p_{w,r,c,V} \ \Rightarrow\ \bigwedge_{n=0}^{|w|-1} g_{r+n,\,c} = w[n]$.

Word boundaries. Words must fit inside the grid and the cells preceding and following must be empty.

$\forall w, r, c.$

$$p_{w,r,c,H} \ \Rightarrow\ \big(c = 1 \vee g_{r,\,c-1} = \bot\big) \ \wedge\ \big(c + |w| - 1 = n \vee (c + |w| - 1 < n \wedge g_{r,\,c+|w|)} = \bot\big) \wedge$$
$$p_{w,r,c,V} \ \Rightarrow\ \big(r = 1 \vee g_{r-1,\,c} = \bot\big) \ \wedge\ \big(r + |w| - 1 = n \vee (r + |w| - 1 < n \wedge g_{r+|w|,c}) = \bot\big)$$

No duplicates. Each word is placed at most once: $\forall\, w_i \in \mathcal{W}\ |\{p_{w,r,c,o} | w = w_i\}| \le 1$.

Every string of letters is a word. Every maximal horizontal (or vertical) non-empty run of length at least two must correspond to exactly one placement.

$$(g_{r,c-1} = \bot \vee c = 1) \wedge g_{r,c} \neq \bot \wedge g_{r,c+1} \neq \bot \implies p_{w,r,c,H} \bigwedge$$
$$(g_{r-1,c} = \bot \vee r = 1) \wedge g_{r,c} \neq \bot \wedge g_{r+1,c} \neq \bot \implies p_{w,r,c,V}$$

All words are connected. Let $C = \{(r, c) \mid g_{r,c} \neq \bot\}$ be the set of non-empty cells. Consider the graph whose vertices are the cells in C, with an edge between two cells whenever they are orthogonally adjacent (horizontally or vertically).

We require this graph to be connected. That is, for every pair of non-empty cells $u, v \in C$, there must exist a sequence of non-empty cells $u = x_0, x_1, \ldots, x_\ell = v$ such that x_i and x_{i+1} are orthogonal neighbours for every $0 \le i < \ell$.

Grid must be non-empty. At least m words must be placed: $|\{p_{w,r,c,o} \,|\, p_{w,r,c,o} = true\}| \ge m$.

3.2 LLM-GUIDED INITIALIZATION

Whilst we can often solve the full crossword SAT instance with an satisfiability solver, this can be computationally expensive, particularly for large grids or high quality thresholds. We augment the

solver with a *warm-start* generated by a large language model (LLM), shown in Algorithm 1. This is broken down into the following steps:

LLM suggestions. The LLM suggests some initial placements of words. Given a word list $\mathcal{W}$, grid size n, and minimum quality m, we prompt an LLM $\mathcal{M}$ to produce a candidate grid of size $n \times n$ that contains at least m words from $\mathcal{W}$. We extract the suggested placements by scanning the LLM's grid for maximal letter runs, considering the horizontal placements first, giving a list of placements $\Pi_{\mathcal{M}}$.

Repair loop. The LLM's output may contain constraint violations (unknown words, out-of-bounds positions, character conflicts). We validate each placement and feed any errors back to the LLM as a repair prompt, iterating up to t times.

Incremental Solving. We greedily filter $\Pi_{\mathcal{M}}$ to retain only valid placements. For each valid suggested placement $\pi = (w, r, c, o) \in valid(\Pi_{\mathcal{M}})$, we construct a conjunction of character-matching constraints, which enforces that each character is placed in the grid:

$$\phi_\pi \;=\; \bigwedge_{\delta=0}^{|w|-1} g_{\mathrm{row}(\pi,\delta),\,\mathrm{col}(\pi,\delta)} = w[\delta], \text{ where } (\mathrm{row}(\pi,\delta), \mathrm{col}(\pi,\delta)) \text{ is the } \delta\text{-th cell of } \pi.$$

These are all added to the SAT encoding, longest word first, and the SAT solver is asked to find a completion for the grid. If the grid cannot be completed, the most recently added constraint is popped from the solver, and solving continues.

Optimization Solver The second approach takes the same set of constraints, but, instead of filtering placements to only valid placements, we add all placements as soft constraints to the solver. Each soft constraint is given a cost, and the solver aims to minimize the cost of the unsatisfied constraints. We assign each suggested placement a weight equal to the length of the word it corresponds to, so that the solver prefers to place longer (and therefore trickier to place) words first. Thus, we delegate filtering the suggested placements to the solver, which can find a valid grid that incorporates as many of the LLM's suggestions as possible, rather than removing invalid placements in the order we discover them. The solver maximizes the total weight of satisfied soft constraints subject to all hard crossword rules being satisfied.

4 Evaluation

We evaluate clue generation against three LLMs, as listed in the results tables. Grid generation experiments were conducted on a 64-bit Ubuntu 22.04.5 LTS system with an Intel Core i9-10900 processor running, with 10 cores and 20 logical threads, and 128 GB of RAM. We use Z3 (de Moura & Bjørner, 2008) version 4.12.2 as the SAT solver.

4.1 Evaluating Cryptic Clue Generation

We evaluate the LLMs' ability to generate clues with and without the DSL and validator loop, on 100 clues per configuration spanning ten clue types. For the generation without DSL, we use a simple prompt which instructs the LLM to come up with a new clue of a specific wordplay type. Outputs are compared against Times Quick Cryptic clues (from 2025/2026) with respect to correctness, clue length, perplexity, semantic distance, and web search hits. A single expert manually evaluated correctness of the clues, and results are reported in Table 4. The DSL plus generator-validator loop improves validity for the weaker models, and, in many cases, generating directly in the DSL already helps before repair is applied. For GPT5.5, correctness is already high and so the loop mostly reduces duplication and increases diversity rather than correctness. On the quality metrics, the generated clues are broadly in the right range but are usually less fluent and less misdirecting than human clues.

A correct clue is not necessarily a good clue if it does not flow well as a sentence, so we also adopt five automated metrics as a proxy for fluidity, drawn from the natural language processing literature and adapted to cryptic crossword clues. We compare the generated clues against the Times Quick Cryptic clues on all these metrics. **Clue length** (in words and characters) verifies that generated clues are not trivially short. **Perplexity** (measured with a pretrained language model) approximates

Algorithm 1 LLM-guided grid placement

Require: word list $\mathcal{W}$, grid size $n = 13$, quality m, max LLM repairs t, use-optimize flag $opt \in \{\mathsf{true}, \mathsf{false}\}$
Ensure: a valid crossword grid G, or FAILURE

$\Phi \leftarrow \textsc{Encode}(\mathcal{W}, n, m)$ ▷ Rules 1–6
$\Pi \leftarrow \textsc{LLMSuggest}(\mathcal{W}, n, m, t)$
if not opt **then** ▷ greedy removal of any invalid entries
  $\Pi \leftarrow \textsc{RemoveInvalid}(\Pi, \mathcal{W}, n)$
  $S \leftarrow \textsc{Solver}(\Phi)$
  for each $\pi = (w, r, c, o) \in \Pi$ **do**
    $S.\textsc{Push}()$ ▷ create a backtrack point
    $\phi_\pi \leftarrow \bigwedge_\delta \; g_{\mathrm{cell}(\pi,\delta)} = w[\delta]$
    $S.\textsc{Add}(\phi_\pi)$
  for each $\pi = (w, r, c, o) \in \Pi$ **do**
    $status \leftarrow S.\textsc{Check}()$
    if $status = \mathsf{SAT}$ **then**
      return $S.\textsc{Model}()$;
    else
      $S.\textsc{Pop}()$
  return None
else
  $S \leftarrow \textsc{OptimizeSolver}(\Phi)$
  for each $\pi = (w, r, c, o) \in \Pi$ **do**
    $\phi_\pi \leftarrow \bigwedge_\delta \; g_{\mathrm{cell}(\pi,\delta)} = w[\delta]$
    $S.\textsc{AddSoft}(\phi_\pi, \; |w|)$ ▷ weight $= |w|$
  $status \leftarrow S.\textsc{Check}()$
  if $status = \mathsf{SAT}$ **then**
    return $S.\textsc{Model}()$;
  else
    return None

the fluency of the surface reading; lower perplexity indicates more natural phrasing. **Semantic distance** between the clue and its answer (computed as $1 -$ cosine similarity of sentence embeddings) quantifies misdirection: high-quality clues should be semantically distant from their answers. Alongside these NLP metrics, we also use the Serper API (Serper, 2026) to search the internet for each clue string. This is to check whether clues were genuinely good or just taken from already published clues on the internet. We apply each metric to human-written Times Quick Cryptic clues and to our generated datasets.

Table 5 shows the scores for the quality metrics described above. The DSL-based system reliably produces valid quick-cryptic-style clues that are similar in basic structure to human-written clues. However, generated clues are typically easier and use less misdirection. This suggests that formal validation addresses structural correctness, but not all aspects of clue quality. Stylistic quality remains partly semantic, subjective, and difficult to capture with a complete formal specification.

Method	Metric	Times 1	Times 2	Gen1	Gen2	Gen3	Gen4
Genxword	Time (s)	1.3	1.3	1.3	1.3	1.3	1.3
	Clues	14.0	13.0	16.0	16.0	14.0	14.0
	Density	46.0	45.0	47.0	47.0	49.0	48.0
SAT Solver	Time (s)	4.8	196.6	77.3	36.6	18.7	126.9
	Clues	**23.0**	22.0	**23.0**	**22.0**	17.0	19.0
	Density	51.0	50.0	53.0	50.0	47.0	52.0
SAT + LLM	Time (s)	25.4	**70.6**	45.9	127.5	29.3	19.7
(3 repairs)	Clues	20.0	21.0	16.0	18.0	18.0	19.0
(optimize solver)	Density	47.3	47.3	47.9	50.5	44.4	48.3
SAT + LLM	Time (s)	98.3	98.6	72.8	51.0	81.0	82.6
(3 repairs)	Clues	**23.0**	**23.0**	18.0	18.0	**23.0**	21.0
(incremental solver)	Density	52.5	52.7	51.3	46.6	52.7	50.5

Table 3: Comparison of grid generation methods across different clue sets from the Times (Times 1-2) and generated by our pipeline (Gen1-4). Clues is the number of clues successfully included in the final grid, and density is proportion of filled squares in the final grid. We omit the stand-alone LLMs, which cannot produce any valid grids.

	GPT4o	GPT5.2	GPT5.5	GPT4o	GPT5.2	GPT5.5	GPT4o	GPT5.2	GPT5.5
	Without DSL			*With DSL*			*With DSL and Validator*		
Correctness	1%	3%	95%	11%	15%	74%	16%	20%	95%
Runtime (s)	208.1	275.2	1320.8	42.5	126.3	310.8	125.6	2141	3257
Num Iterations	1	1	1	1	1	1	2.97	2.79	1.25
Unique Answers	100	99	70	9	14	67	42	65	84

Table 4: Correctness results without DSL, with DSL, and with DSL and validator.

	GPT4o			*GPT5.2*			*GPT5.5*			*Times*
	w/o DSL	DSL	DSL+Validator	w/o DSL	DSL	DSL+Validator	w/o DSL	DSL	DSL+Validator	Times
Length (chars)	32	21.3	25.0	35.3	29.1	29.5	33.5	32.1	32.8	37.59
Length (words)	5.00	3.64	4.06	6.00	4.53	4.70	5.28	4.84	4.95	5.90
Perplexity (distilgpt2)	1484	13443.8	12149	1102	9152.5	9520	36378	184501.9	192698	12483
Perplexity (gpt2)	989	4714.9	4978	2186	3511.0	3840	16177	18207.3	22246	4932
Perplexity (gpt2xl)	1737	2493.0	2545	1102	2950.9	3453	18870	35900.6	36116	4572
Semantic Dist.	0.59	0.661	0.671	1.00	0.681	0.689	0.70	0.702	0.711	0.80
# Found Online	0	4	5	0	2	2	5	10	10	-

Table 5: Average quality results for correct clues only, with Times clues included for reference.

4.2 Evaluating Grid Generation

We evaluate grid generation on six sets of 50 answers: two sets drawn from The Times and four sets generated by our clue-generation pipeline. We compare four approaches: direct LLM generation, the heuristic crossword generator Genxword, the SAT encoding described in Section 3.1, and two variants of our LLM-guided SAT method. For the direct-LLM baseline, the model is given a set of 50 answers and asked to construct a valid 13×13 grid containing as many of them as possible. Each LLM experiment is repeated three times and the results are averaged. Genxword greedily places words at valid intersections, repeating the process from different initial placements and retaining the best grid found within its search budget.

For the SAT-based methods, we initially require at least 22 answers to be placed. If no solution is found within 200 seconds, we reduce this requirement using binary search until a satisfiable grid is obtained. The LLM-guided variants use GPT-4o to propose initial placements. In the optimization variant, these suggestions are added as weighted soft constraints; in the incremental variant, suggested placements are added as constraints and removed when they prevent the solver from finding a completion. In both cases, the underlying crossword constraints remain hard constraints, so any returned grid is valid by construction.

Table 3 summarizes the results. Direct LLM generation fails to produce a valid grid on any of the six answer sets, averaging more than 15 placement errors per grid. All symbolic methods return valid grids. Genxword is fastest at about 1.3 seconds per instance, but places only 13–16 answers, while standalone SAT places 17–23 answers at 47–53

The LLM-guided results show that model suggestions can alter the solver's search trajectory, but do not provide a uniform speed-up. The optimization variant reduces mean runtime from 76.8 to 53.1 seconds, with large gains on Times 2 and Gen4, but is slower on easier instances and places fewer answers on average (18.7 vs. 21.0). The incremental variant preserves the standalone SAT average of 21.0 answers and can produce denser grids, but likewise has mixed effects on runtime.

These results suggest that the main benefit of LLM guidance is instance-dependent rather than universal: When the standalone SAT problem is difficult, useful placement suggestions can substantially reduce search time or steer the solver toward a denser solution. When the SAT instance is already easy, the overhead or additional constraints introduced by the LLM can instead make solving slower. The two integration strategies also expose different trade-offs: soft optimization tends to favor lower runtime at some cost to the number of answers placed, whereas incremental placement can preserve or improve grid density but does not consistently reduce runtime.

4.3 Limitations of the Validator

It is impossible to create a sound and complete validator for a problem which inherently involves reasoning about natural language, which is ambiguous and evolving: even expert human setters sometimes disagree on whether a clue is valid. Our validator also utilises fall back calls to an LLM, for example for abbreviations, which cannot be guaranteed to be accurate. However, we have evaluated all the clues in the evaluation by hand as well as with the validator, and so we are confident that the results reported here are accurate. Our results demonstrate that the validator supplies a sufficiently precise feedback signal to improve the accuracy of generation.

5 Related Work

Crossword Solving. Standard crossword solving is relatively mature: systems such as Proverb, Dr. Fill, and the Berkeley Crossword Solver combine clue-answer retrieval with global constraint reasoning and achieve near-expert performance (Littman et al., 2002; Ginsberg, 2011; Wallace et al., 2022). Recent LLM-based variants improve this further when paired with structured search or logic constraints (Tao, 2024; Saha et al., 2025).

Cryptic solving remains much harder. Early grammar-based approaches captured clue structure but scaled poorly (Williams & Woodhead, 1979; Deits, 2015). Neural methods improved accuracy on Cryptonite but remain well below human performance (Efrat et al., 2021; Rozner et al., 2021; Sadallah et al., 2025; Friedlander & Fine, 2009). Andrews & Witteveen (2025a) uses executable DSL proofs to verify proposed wordplay explanations, which our work extends from solving to generation. In particular, this is the closest clue-level approach to ours: we extend the same DSL from answer-verification in solving to full clue generation with iterative repair.

Crossword Generation. Most generation work targets standard crosswords: clue text is extracted from corpora and grids are filled using heuristic search or CSP methods (Rigutini et al., 2012; 2008; Ranaivo-Malançon et al., 2013; Esteche & Romero, 2017; Beacham et al., 2001; Agarwal & Joshi, 2020). Grid construction itself is NP-complete (Garey & Johnson, 1990). In contrast, cryptic clue generation has seen little progress beyond early template-based systems (Hardcastle, 2007). In those rule-based systems, creativity is inherently bounded by the fixed hand-written rules and dictionaries available to the generator. Although there is extensive work on SMT/CSP-style grid layout, we are not aware of prior work that uses LLM suggestions to warm-start symbolic crossword grid solving, nor of any system that performs full end-to-end cryptic crossword generation.

Constrained and Neurosymbolic Generation. Constrained decoding methods enforce formal syntax at generation time, e.g., grammar- or automata-based token filtering (Scholak et al., 2021; Zhang et al., 2024), while post-hoc methods iteratively repair outputs (Madaan et al., 2023). These methods are effective when constraints are mostly local and syntactic, but do not capture the combination of global grid consistency, letter-level transformations, and semantic clue-definition agreement required in our setting. Hybrid approaches that couple LLMs with formal representations and solver-based checking are closest to our setting (Mora et al., 2024). This work also finds that requiring the model to generate in a concise DSL improves performance even before validation or repair is applied. Our contribution applies this neurosymbolic pattern to full cryptic crossword generation.

6 Conclusion

LLMs alone cannot reliably generate cryptic crosswords: weaker models produce valid clues only 1–3% of the time, and no model tested ever produced a valid 13×13 grid. Combining LLMs with a DSL-based validator and a satisfiability solver closes both gaps—raising clue validity without duplication and generating valid grids on every word set tested. Generated clues match human clues in length and semantic distance, but trail in fluency and misdirection, suggesting that formal validation is necessary but not sufficient for high-quality generation. Our future work will look at using the quality metrics as direct feedback within the validation loop to improve the quality of generated clues.

7 STATEMENT OF AI USE

We used generative AI to assist with formatting of tables, as well as grammar and spelling checks in the writing. We also used coding agents to assist in writing code for the implementation of our approach. We manually checked all LLM-generated text and code, and take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Charu Agarwal and Rushikesh K. Joshi. Automation Strategies for Unconstrained Crossword Puzzle Generation, July 2020. URL http://arxiv.org/abs/2007.04663. arXiv:2007.04663 [cs].

Martin Andrews and Sam Witteveen. Generating Code to Verify Cryptic Crossword Reasoning. March 2025a. URL https://openreview.net/forum?id=2nC7zy7adD.

Martin Andrews and Sam Witteveen. A Reasoning-Based Approach to Cryptic Crossword Clue Solving, June 2025b. URL http://arxiv.org/abs/2506.04824. arXiv:2506.04824 [cs].

Adam Beacham, Xinguang Chen, Jonathan Sillito, and Peter van Beek. Constraint Programming Lessons Learned from Crossword Puzzles. In *Proceedings of the 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence*, AI '01, pp. 78–87, Berlin, Heidelberg, June 2001. Springer-Verlag. ISBN 978-3-540-42144-3.

Ross Beresford and Mark Longair. cryptic-crossword-indicators-and-abbreviations. https://github.com/mhl/cryptic-crossword-indicators-and-abbreviations, 1992–present. GitHub repository; original list compiled by Ross Beresford, maintained by Mark Longair.

Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof (eds.), *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pp. 337–340. Springer, 2008. doi: 10.1007/978-3-540-78800-3_24. URL https://doi.org/10.1007/978-3-540-78800-3_24.

Robin Deits. rdeits/cryptics, 2015. URL https://github.com/rdeits/cryptics. original-date: 2012-11-03T19:38:02Z.

Robin Deits. CrypticCrosswords.jl: Cryptic crossword clue solver written in julia. https://github.com/rdeits/CrypticCrosswords.jl, 2018. GitHub repository, accessed 2026-02-19.

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

Juan Esteche and Roberto Romero. Clue-instruct: Text-based clue generation for educational crossword puzzles. In *Conference proceedings*, 2017.

Kathryn Friedlander and Philip Fine. Expertise in cryptic crossword performance: An exploratory survey. January 2009. URL https://www.academia.edu/3733749/Expertise_in_cryptic_crossword_performance_An_exploratory_survey.

Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, October 1990. ISBN 978-0-7167-1045-5.

Matthew L. Ginsberg. Dr.fill: crosswords and an implemented solver for singly weighted csps. *J. Artif. Int. Res.*, 42(1):851–886, September 2011. ISSN 1076-9757.

David Hardcastle. Cryptic Crossword Clues: Generating Text with a Hidden Meaning. In Stephan Busemann (ed.), *Proceedings of the Eleventh European Workshop on Natural Language Generation (ENLG 07)*, pp. 147–150, Saarbrücken, Germany, June 2007. DFKI GmbH. URL https://aclanthology.org/W07-2323/.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In Mirella Lapata, Phil Blunsom, and Alexander Koller (eds.), *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 427–431, Valencia, Spain, April 2017. Association for Computational Linguistics. URL https://aclanthology.org/E17-2068/.

Michael L. Littman, Greg A. Keim, and Noam Shazeer. A probabilistic approach to solving crossword puzzles. *Artificial Intelligence*, 134(1):23–55, January 2002. ISSN 0004-3702. doi: 10.1016/S0004-3702(01)00114-X. URL https://www.sciencedirect.com/science/article/pii/S000437020100114X.

Derrick Somerset Macnutt. *Ximenes on the Art of the Crossword*. Methuen, London, 1966. Written under the pseudonym Ximenes.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: iterative refinement with self-feedback. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Federico Mora, Justin Wong, Haley Lepe, Sahil Bhatia, Karim Elmaaroufi, George Varghese, Joseph E. González, Elizabeth Polgreen, and Sanjit A. Seshia. Synthetic programming elicitation for text-to-code in very low-resource programming and formal languages. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

B. Ranaivo-Malançon, T. Lim, J. Minoi, and A. Jupit. Automatic generation of fill-in clues and answers from raw texts for crosswords. In *8th International Conference on Information Technology in Asia (CITA)*, pp. 1–5, 2013.

Leonardo Rigutini, Michelangelo Diligenti, Marco Maggini, and Marco Gori. A Fully Automatic Crossword Generator. In *2008 Seventh International Conference on Machine Learning and Applications*, pp. 362–367, December 2008. doi: 10.1109/ICMLA.2008.104. URL https://ieeexplore.ieee.org/abstract/document/4724999.

Leonardo Rigutini, Michelangelo Diligenti, Marco Maggini, and Marco Gori. Automatic generation of crossword puzzles. *International Journal on Artificial Intelligence Tools*, 21(03): 1250014, June 2012. ISSN 0218-2130. doi: 10.1142/S0218213012500145. URL https://www.worldscientific.com/doi/abs/10.1142/S0218213012500145. Publisher: World Scientific Publishing Co.

Joshua Rozner, Christopher Potts, and Kyle Mahowald. Decrypting cryptic crosswords: Semantically complex wordplay puzzles as a target for NLP. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=136ihvjd0sJ.

Abdelrahman Sadallah, Daria Kotova, and Ekaterina Kochmar. Are LLMs Good Cryptic Crossword Solvers?, January 2025. URL http://arxiv.org/abs/2403.12094. arXiv:2403.12094 [cs].

Soumadeep Saha, Sutanoya Chakraborty, Saptarshi Saha, and Utpal Garain. Language models are crossword solvers. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 2074–2090, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.104. URL https://aclanthology.org/2025.naacl-long.104/.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9895–9901, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.779. URL https://aclanthology.org/2021.emnlp-main.779/.

Serper. Serper api, 2026. URL https://serper.dev.

Hillary Tao. Automated crossword solver using GPT-4 and logic constraints, 2024. URL https://escholarship.mcgill.ca/concern/theses/9306t472j. Publisher: McGill University.

Eric Wallace, Nicholas Tomlin, Albert Xu, Kevin Yang, Eshaan Pathak, Matthew Ginsberg, and Dan Klein. Automated crossword solving. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3073–3085, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.219. URL https://aclanthology.org/2022.acl-long.219/.

P. W. Williams and D. Woodhead. Computer assisted analysis of cryptic crosswords. *The Computer Journal*, 22(1):67–70, January 1979. ISSN 0010-4620. doi: 10.1093/comjnl/22.1.67. URL https://doi.org/10.1093/comjnl/22.1.67.

Honghua Zhang, Po-Nien Kung, Masahiro Yoshida, Guy Van den Broeck, and Nanyun Peng. Adaptable logical control for large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 115563–115587. Curran Associates, Inc., 2024. doi: 10.52202/079017-3670. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d15c16cf5619a2b1606da5fc88e3f1a9-Paper-Conference.pdf.

A Full Prompts

This section contains prompts omitted from the main paper.

A.1 Generation Prompt (for anagram example)

You are a cryptic crossword compiler and formal verifier.

Your task is to generate an original cryptic crossword clue (of a anagram type) together with a FORMAL, STRUCTURED PROOF of the wordplay.

Available Proof Functions / Concepts

The following are the ONLY allowed semantic operations in the proof.
Do not invent new operations.

```python
Action = Enum('Action', 'ANAGRAM,REMOVE_FIRST,INITIALS,REMOVE_LAST,GOES_INSIDE,
GOES_OUTSIDE,REVERSE,SUBSTRING,GOES_NEXT_TO,REMOVE_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'
  return True # defined elsewhere
def is_abbreviation(phrase:str, test_abbreviation:str) -> bool:
  # Determines whether 'test_abbreviation' is a valid abbreviation or short form for 'phrase'
  return True # defined elsewhere
def action_type(phrase:str, action:Action) -> bool:
```

```
  # Determines whether 'phrase' might signify the given 'action'
  return True # defined elsewhere
def is_anagram(letters:str, word:str) -> bool:
  # Determines whether 'word' can be formed from 'letters' (i.e. is an anagram)
  return True # defined elsewhere
def is_homophone(phrase:str, test_homophone:str) -> bool:
  # Determines whether 'test_homophone' sounds like 'phrase'
  return True # defined elsewhere
```

All proof steps must correspond to these operations.

Clue Requirements

- Use British cryptic crossword conventions.
- The clue must contain EXACTLY ONE definition.
- The surface reading must be natural English.
- The answer length must match the pattern.

Required Output Format (MANDATORY)

The contents must follow this structure:

````
```python
def proof(answer="DECIMAL", clue="the point of medical treatment", pattern='7'):
  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")
```
````

Constraints

- You MUST use a ```python``` block, with both opening and closing backticks
- Use ONLY operations specified above.
- Do NOT use any variables. Each line of code must be independent from one another.
- Do NOT use any curly braces to indiciate definition words.
- Do NOT include any text outside the multiline string.
- Do NOT omit intermediate steps.

Task

Generate one original cryptic crossword clue (of a anagram type)
and its complete structured proof that satisfies all of the above constraints.

Output must start with exactly three backticks, the word python, then a newline.
End the output with exactly three backticks on a line by themselves.
Do not omit these backticks under any circumstances.

A.2 LLM GRID GENERATION PROMPT

You are an expert crossword constructor.

The following are crossword answers with their clues.

Your task is to reconstruct a valid crossword layout.

Requirements:
– Select as many answers from the provided list of clues as you can.
– Every selected answer appears exactly once.
– Every answer is placed either horizontally or vertically.
– Words intersect only on identical letters.
– Every intersection must belong to both words.
– No adjacent words may touch unless they intersect.
– The entire crossword must fit inside a 13x13 grid.
– Use '#' for black squares.
– Use uppercase letters only.
– The grid must be rectangular (13 strings of length 13).

Ensure that you return a VALID crossword grid that conforms to these requirements.

Return ONLY the grid. in this format(with no Markdow, JSON, or explanations):

Example format:

OBSERVE#MATCH
P#Y#E#N###R##
E#R#L#AWKWARD
R#U#E#M###M##
APPEARED#SPED
T###S#L#G#L#E
ISRAEL#REPEAT
O#E#D#N#N###E
N#C##REMEMBER
##Y#K#C#R#R#G
ANCIENT#O#I#E
##L#P#A#U#N#N
SHEET#RESPECT

Clues:

{clues}
""""

B ADDITIONAL EXPERIMENTAL RESULTS

B.1 INVALID GRID GENERATION

Although GPT5.5 did not place any incorrect words within its generated grids, it produced grid that were invalid through being unconnected. An example is shown below:

MORAL#SKY#SAP
###B#########
BALL#LIE#INCH
###E#########
MOW#LONE#AGED
#############
FOYER#OTHER##
#############
STAFF#MINOR##

Method	Metric	Times 1	Times 2	GPT 1	GPT 2	GPT 3	GPT 4
GPT4o	13x13	0/3	0/3	0/3	0/3	0/3	0/3
	Time (s)	2.90	2.25	2.55	3.02	3.37	3.07
	Clues	31	35	36	33	36	39
	Invalid	22	23	24	24	26	31
	Density (%)	38.68	46.10	43.56	43.28	44.83	41.46
GPT5.2	13x13	0/3	2/3	1/3	1/3	1/3	1/3
	Time (s)	3.75	6.14	8.33	5.97	7.75	3.44
	Clues	14	14	20	24	16	13
	Invalid	8.67	6.67	10.33	11.67	8.67	3.67
	Density (%)	25.64	26.70	42.41	57.02	36.03	36.20
GPT5.5	13x13	3/3	3/3	3/3	3/3	3/3	3/3
	Time (s)	152.44	181.66	197.38	212.26	200.26	243.58
	Clues	18	18	14	15	15	15
	Invalid	0	0	0	0	0	0
	Density (%)	47.53	44.77	44.77	48.32	45.37	46.55
Genxword	Time (s)	1.3	1.3	1.3	1.3	1.3	1.3
	Clues	14	13	16	16	14	14
	Density (%)	46	45	47	47	49	48
SAT Solver	Time (s)	4.76	196.64	77.31	36.61	18.70	126.90
	Clues	**23**	22	**23**	**22**	17	19
	Density (%)	51	50	53	50	47	52
SAT + LLM (1 repair, optimized)	Time (s)	65.93	110.71	77.77	100.96	79.81	145.25
	Clues	20	22	17	18	22	20
	Density (%)	49.31	49.11	47.54	47.53	51.68	47.14
SAT + LLM (3 repairs, optimized)	Time (s)	25.36	**70.62**	45.86	127.50	29.32	19.73
	Clues	20	21	16	18	18	19
	Density (%)	47.34	47.34	47.93	50.49	44.38	48.32
SAT + LLM (1 repair, incremental)	Time (s)	68.45	132.96	105.88	77.59	126.65	91.33
	Clues	21	22	17	18	**23**	21
	Density (%)	48.52	53.26	48.52	49.70	51.48	50.89
SAT + LLM ((3 repairs, incremental))	Time (s)	98.29	98.56	72.75	51.03	80.95	82.58
	Clues	**23**	**23**	18	18	**23**	21
	Density (%)	52.47	52.66	51.28	46.55	52.66	50.49

Table 6: Full grid generation results. Standalone LLMs and SAT+LLM configurations run three times and averaged. Genxword and SAT solver run just once. 13x13 refers to whether produced grids were of the specified 13x13 dimensions. Time refers to how long the configuration took to generate the grid. Clues refers to the number of clues taken from the clue file that the configuration managed to place in the grid. Invalid refers to the number of words not found in the clue file that the configuration placed in the grid (including words created by invalid touching of valid words). Density refers to the percentage of grid cells filled with characters as opposed to being blank. Note: Although GPT5.5 did not place any incorrect words, it produced grid that were invalid through being unconnected. An example is within the supplementary materials.

##G##########
GORGE#PINCH##
##E##########
SCENE#ROUND##