An Industrial Crossword Generator Driven by a Local Language Model

Shyamal Suhana Chandra
Domino Data Systems

2026

Abstract

This paper describes a production-style command-line tool that turns a topic into a printable crossword. A local Ollama model writes short questions whose answers are single words. A Rust placer lays those words on a square grid, and the tool writes a PDF containing the blank puzzle, numbered clues, and a filled answer key. The design favors a local model, bounded requests, and a deterministic seed so the same inputs can be reproduced. The system is aimed at operators who need themed puzzles without sending topic text to a hosted model API.

1 Introduction

Themed crosswords are a recurring content task: training handouts, event programs, and classroom exercises all need a grid, clues, and an answer key. Authoring them by hand is slow, and hosted language-model APIs are a poor fit when the topic is internal or the machine must work offline.

The tool in this paper is a single Rust binary, `crossword`. The operator names a topic. The program asks a local Ollama server for question-and-answer pairs, places as many answers as the grid rules allow, and writes a PDF. Figure-level workflow is linear: generate candidates, place words, render pages.

2 Requirements

The industrial constraints that shaped the tool are:

- The published command is `crossword` on ARM64 macOS. Ollama runs on that machine. The server is `http://127.0.0.1:11434`. These instructions pass `--model gpt-oss:20b`. Omitting `--model` selects `llama3.2`.
- The grid is square. The default is 25×25. The operator may choose any size from 5 to 40.
- Answers that appear in the grid are single words of letters A–Z, three to twelve characters long, and never longer than the grid.
- Clues are short questions. The answer must not appear in the clue.
- The PDF contains a blank numbered grid, Across and Down clues, and a filled answer key.
- A seed makes placement reproducible. If the operator omits it, the program prints the seed it used.

- A large candidate request must not depend on one long model response. Entries are requested in small batches.

3 System

3.1 Candidate generation

The generator posts to Ollama's `/api/chat` endpoint with JSON mode enabled. Each request asks for at most eight entries. The prompt requires a JSON object whose `entries` array holds `answer` and `clue` strings. The client keeps a read timeout of three minutes per batch, retries a failed batch once, and stops after two batches that add no new words. Later batches are told not to reuse answers already accepted. Answers that contain spaces, digits, or punctuation are discarded. Duplicate answers are discarded. Clues that contain the answer as a word are discarded.

If the named model is missing, the client reads Ollama's error text. When that text says the model was not found, it suggests an installed model whose name is within a small edit distance.

3.2 Placement

Words are shuffled with the seed and then ordered so longer words are considered first. The longest word is placed across the middle of the grid. Each later word must cross at least one existing letter, must not run along an existing word, and must not sit beside another word in the perpendicular direction except at a crossing. The search for the next word scores every remaining candidate and keeps the placement with the most crossings. Tied scores prefer the earlier candidate, then Across, then the uppermost and leftmost cell. The search runs in parallel across candidates. The program refuses a grid with fewer than eight placed words.

3.3 PDF

The document is US Letter. The first page is the blank puzzle: black squares where no letter was placed, and a number in the corner of each word start. Following pages list Across clues and then Down clues. The last page repeats the grid with letters filled in. Every page carries the footer "AI Crossword Puzzle Generator, Domino Data Systems by Shyamal Suhana Chandra, Copyright (C) 2026".

4 Operator interface

Release v0.1.0 is only for ARM64 on macOS. The operator installs the public tap, then runs the `crossword` command. No source checkout is required.

```
brew trust --tap Domino-Data-Systems/crossword
brew tap Domino-Data-Systems/crossword
brew install crossword
crossword --topic "the solar system" --model gpt-oss:20b --verbose
```

The topic is required. Size, output path, model, host, candidate count, seed, and a verbose progress switch are optional. Ollama is the macOS menu-bar app, left running so it serves the local daemon. The model `gpt-oss:20b` is pulled before the command above. Verbose mode draws a progress bar on standard error while batches return and while words are placed. A normal run ends with one line naming the PDF, the grid size, the number of words, and the seed.

5 Operational notes

Requesting hundreds of entries from a large local model is dominated by generation time, not by placement. Small batches keep each HTTP response inside the read timeout and let the progress bar advance after every batch. Placement searches the remaining words in parallel. The grid that results is a legal crossword under the placer's rules, but it is not a fully checked American newspaper grid: many letters are unchecked, and the black-square pattern is not required to be symmetric.

Model quality dominates puzzle quality. A small model may invent words or write questions that do not match the answer. The filters remove malformed tokens. They do not check that a clue is factually correct.

6 Conclusion

The tool is a local pipeline from topic to printable crossword. Ollama supplies themed question-and-answer pairs. A seeded Rust placer builds the grid. The PDF is ready to print, with clues and an answer key. Operators install one command, pass a topic and `--model gpt-oss:20b`, and can rerun a puzzle from the printed seed.