CrosswordLLM: A Benchmark Framework for Evaluating Large Language Models on Linguistic Reasoning in Crossword Puzzles

Alex Frugé

May 2, 2025

Abstract

I'll fill this out later once I get some results, and finalize any fine-tuning steps I plan on making (embeddings, prompt engineering, etc).

Introduction

Large Language Models (LLMs) have demonstrated remarkable proficiency in natural language understanding and generation. However, their ability to solve complex linguistic tasks, such as crosswords, remains an open area of investigation. This project aims to design an evaluation pipeline to test how well different LLMs can solve crossword clues. By analyzing performance variations across model sizes and architectures, I seek to understand whether larger models inherently perform better, or if architectural differences contribute more significantly to success. Additionally, I will explore how contextual information, prompt engineering, and explicit length requirements for answers may affect model performance. This research will not only offer insights into the cognitive capabilities of LLMs but also contribute to broader AI evaluations in natural language processing (NLP).

Literature Review

Foundational Work

The sources within the Foundational Work section mainly go discuss how LLMs scale better with more compute, which was one of the attributes I wanted to verify in my crossword case.

- Brown et al. (2020) introduced GPT-3, demonstrating that large-scale LLMs can perform a variety of tasks with minimal examples, a technique known as few-shot learning. This work established the potential of LLMs to generalize across tasks without extensive fine-tuning.
- Chowdhery et al. (2022) presented PaLM, a 540-billion-parameter model trained using the Pathways system. PaLM achieved state-of-the-art results on numerous benchmarks, highlighting the benefits of scaling model size and training data for improved performance.

Enhancing Few-Shot Learning and Prompting Techniques

This section highlights methods for improving language model performance with limited data through better prompt design and reasoning strategies.

- Gao et al. (2020) proposed LM-BFF, a method combining prompt-based fine-tuning with automated prompt generation to improve few-shot learning in smaller models. This approach is particularly relevant for scenarios with limited annotated data, such as specialized crossword clues.
- Wei et al. (2022) introduced Chain-of-Thought (CoT) prompting, where models generate intermediate reasoning steps before arriving at an answer. CoT prompting significantly improved performance on complex reasoning tasks, suggesting its applicability to the nuanced reasoning required in crossword solving. I ended up using Chain of Thought prompting as one of my enhancements to the existing models.
- Zhou et al. (2023) developed Automatic Prompt Engineer (APE), a system that automatically generates and selects effective prompts. APE outperformed human-crafted prompts on various tasks, indicating the potential for automated prompt optimization in crossword applications.

Previous Applications to Crossword Solving

Saha et al. (2024) demonstrated that current LLMs exhibit significant competence in solving cryptic crossword clues, outperforming previous state-of-the-art

results by a factor of 2–3. They developed a search algorithm enabling LLMs to solve full crossword grids, achieving 93% accuracy on New York Times puzzles. This work underscores the feasibility of applying LLMs to complex language tasks like crosswords

Emerging Techniques and Considerations

Pattern-Aware Chain-of-Thought Prompting (Zhang et al., 2024) emphasizes the importance of diverse reasoning patterns in prompts to enhance model generalization and robustness. This approach could further improve LLM performance on varied crossword clue types.

Future Direction

Integrating these advancements suggests a multifaceted approach to developing an LLM-based crossword-solving suite:

- Model Selection: Utilize large-scale models like PaLM for their superior reasoning capabilities.
- Few-Shot Learning: Implement LM-BFF techniques to fine-tune models with limited crossword-specific data.
- Reasoning Enhancement: Apply CoT and pattern-aware prompting to improve the model's ability to handle complex clues.

By building upon these foundational and contemporary works, the proposed suite aims to advance the capability of LLMs in solving crossword puzzles with high accuracy and efficiency.

Methodology

Data Sourcing & Explanation

The data for this project was pulled from Darin Hawley's New York Times Crossword Clues & Answers 1993-2021 dataset. (Hawley, 2021) This dataset contains 781,573 entries, each of which come with the date the puzzle containing the clue was published, the clue itself, and the answer.

The NYT dataset is being used as our training data, as it contains a good spread of clue variety, including some pop-culture related clues, some wordplay, and a wide variety of difficulty in the clues themselves. It also contains a good number of clues involving multiple words in a clue being concatenated into one continuous word (see Table 1 for some examples of this). I'll also use the NYT dataset for a testing suite that tests the selected models on more conventional clues.

Answer	Clue
EATAT	Irritate
ALQAEDA	War on terror target
ASGOODASGOLD	100% reliable

Table 1: *Example clues from NYT Crossword Clues.*

Libraries and Tools Used

The core libraries used within this package are **PyTorch 2.0** (training original models), **TransformerLens** (loading original models), **HuggingFace Transformers** (loading tokenizers), as well as **Pandas** and **Scikit-learn**.

The system leverages the **Pythia** suite of autoregressive language models (Biderman et al., 2023) as its primary base architecture due to their transparency, reproducibility, and controlled scaling properties. Pythia models provide a standardized framework for experimentation, with versions ranging from 70M to 12B parameters, allowing systematic comparisons across model sizes. For the sake of the models being usable on my computer, the largest model I tested with contained 1.3 billion parameters.

Training Process

The training process follows a structured pipeline designed to optimize the model's ability to generate accurate crossword answers while incorporating task-specific enhancements. Below is a high-level overview of the training workflow.

The system ingests a structured dataset of (clue, answer, length) tuples, derived from the NYT Crossword Clue Dataset. The clues are then standardized (removing metadata like enumeration), and answers are normalized to uppercase alphabetic strings. Answer lengths are explicitly annotated, either as single integers (for single-word answers) or comma-separated values (for multi-word answers). Data is partitioned into training (80%) and evaluation (20%) sets, ensuring no overlap.

Clues are formatted into instructional prompts (e.g., "Solve the crossword clue: [CLUE] ([LENGTH]). Answer:") to guide the model's generation. Multiple prompting strategies (basic, detailed, chain-of-thought) are explored to improve task alignment. Inputs are tokenized using the base model's pretrained tokenizer, with padding/truncation to a fixed sequence length.

Once all of the data is brought in and cleaned up, the user specifies which model they want to use as a base, as well as which enhancement they want to add to that base via a CLI (e.g. main.py –model EleutherAI/pythia-70m –enhancement length_aware

would train the pythia-70m model with the length aware enhancement added on).

The model is trained via causal language modeling, where it predicts the answer tokens autoregressively. The loss is computed only on answer tokens (excluding the prompt). Depending on which enhancements the user specifies, there could be intervention here by said enhancements.

The training loop itself uses the AdamW optimizer with a learning rate of $5E-05$, batch size of 32, and gradient clipping, as well as dropout (where applicable) and early stopping to prevent overfitting.

Enhancements

The base autoregressive language model is augmented with several task-specific enhancements designed to improve its crossword-solving capabilities. These modifications target key challenges in clue-answer generation, including length conditioning, semantic grounding, and controlled decoding.

Length-Aware Modeling One current problem with standard LLMs is that they struggle to generate answers of exact lengths (e.g., 5 letters for "Capital of France").

One solution to this problem is to add explicit length conditioning via **length embeddings** and **positional bias**.

Length embeddings are learned embedding layers which map target lengths (integers) to vectors, which are then added to the LM's hidden states before the output layer, which are then used to select length-specific output heads (one per possible length).

Positional bias is another position embedding which encourages the model to stop early when the target length is reached (via EOS token bias), and to avoid over-generation (suppression of non-alphabetic tokens beyond the target length).

Contextual Embedding Fusion LMs may lack deep semantic understanding of clues (e.g., parsing wordplay like "Composer of 'The Planets' (5)" $\rightarrow$ "HOLST").

The solution that I came up with is a hybrid architecture combining a **frozen sentence embedder** and **gated fusion**.

The frozen sentence embedder is derived from a pretrained model (e.g., all-MiniLM-L6-v2) encodes clues into dense vectors, capturing synonym relationships ("composer" $\rightarrow$ "musician") and entity linkages ("The Planets" $\rightarrow$ "Gustav Holst").

The LM's hidden states are combined with projected embeddings via:

$$\mathbf{h}_{\text{fused}} = \sigma\left(\mathbf{W}_g\left[\mathbf{h}_{\text{LM}}; \mathbf{h}_{\text{emb}}\right]\right) \odot \mathbf{h}_{\text{LM}} + \left(1 - \sigma\left(\mathbf{W}_g\left[\mathbf{h}_{\text{LM}}; \mathbf{h}_{\text{emb}}\right]\right)\right) \odot \mathbf{h}_{\text{emb}}$$

where $\mathbf{W}_g$ is a learned gate matrix.

Testing Process

Results

Model	Accuracy	Length Match Accuracy
pythia-14m	0.0000	0.0320
pythia-31m	0.0000	0.2580
pythia-70m	0.0000	0.1210
pythia-160m	0.0030	0.2150
pythia-410m	0.0230	0.2400
pythia-1.3b	0.0270	0.1970

Table 2: *First run across Pythia Suite using NYT Data (n=12,800, batch size=32, epochs=5)*

Referencing a table using its label: Table **??**.

Conclusion