Hybrid Hangman

Data Collection & Cleaning :

We began by pulling together two large sources: Dolph's open-source word list on GitHub and our own 250 000-word training file. After concatenating and lowercasing everything, we noticed entries like **Jean-Christophe**, so we split those into `jean`, `christophe` **and** `jeanchristophe` (to cover both components and the compound). Finally, we dropped any entries with non-alphabetic characters (so no stray digits or symbols) and deduplicated the lot, leaving us with roughly 450 000 clean, unique words.

Sample Generation for the Neural Net :

Rather than hand-label data, we let the computer simulate millions of Hangman games: for each word (or, for very long words, random 10-letter windows), we played out up to seven entropy-based splits and recorded triples like :

```
(pattern: "_ e _ l o _ _",
 guessed: ['e','o'],
 next: 'r')
```

By weighting our samples so each word-length got its fair share (3 million total), we ensured the net sees short and long words equally. This pipeline also pads patterns to 20 characters and encodes the "already guessed" mask as a 26-bit vector.

Neural-Net Training & Limits :

Our model is a straightforward 4-layer MLP (512→512→256→26) with dropout 0.25, trained for eight epochs under Adam (lr=1 e-3). On held-out samples it tops out around 85 % accuracy ; but it learns to **maximize information gain**, not necessarily to "hit" the most likely letter. In practice it can struggle on words like **neurohumours**, where the pattern `_ _ r s` should resolve to "ours" but the entropy heuristic still chases splits instead of the obvious vowel.

Why Ensembling? :

Why an ensemble, and not just one or two methods?

Frequency-only

• *Strength:* First two guesses eliminate ~50 % of words by hitting the most common letters for that length.

• *Weakness:* With only those letters, the remaining pool is still too large for reliable end-game picks, e.g. after "E, A" on a 10-letter word, hundreds of candidates remain, and a single high-frequency letter like "T" may not actually appear in your specific word.

Neural-net-only

• *Strength:* It learns to ask high-information questions based on the current pattern.

• *Weakness:* It's trained to "split" the pool, not to "complete" the word, so late in the game it still suggests letters that divide the small set rather than fill the blanks. For instance, at `_ _ R S`, the net might pick "T" or "N" instead of the guaranteed "O."

Regex-only

• *Strength:* Once you know enough letters, pattern-matching finishes words with 100 % precision.

• *Weakness:* It needs at least ~5/6 letters revealed to narrow the list to a handful; before then, applying `^...rs$` against hundreds of thousands of words is too noisy and returns too many matches to be decisive.

Frequency + Neural-net

You still lack a reliable end-game; after four moves, the net's entropy objective can pull you away from the single correct completion.

Neural-net + Regex

Without smart openers, your first two moves leave you with near-random patterns that confuse the net and delay reaching the five-letter threshold for regex.

Frequency + Regex

Two common letters can't reveal enough of the board to make regex matching effective until much later, costing several extra guesses.

Ensemble Strategy & Regex-Driven End-Game :

To bridge that gap, we layered three phases:

- **Length-specific openers (moves 1–2)**: we simply play the top two letters for that exact word length (so "e,a" for 5-letter, "s,t" for 8-letter).
- **Neural-net picks (moves 3–5/6)**: we feed the partially filled pattern plus guessed-mask into the trained MLP and take its two highest-scoring outputs.
- **Regex-based end-game heuristic (move 6/7→finish)**: once you have revealed at least five letters, the list of possible words is small enough to filter directly. You take the current board pattern; for example, `_ _ R S` and turn it into a regular expression (`^..rs$`). You match that against your full word list, immediately picking up endings like **ours** in "colours" or "humours". If more than one word still fits, you then apply the same expression to selected portions of each candidate: first half-word slices, then one-third-word slices (for example, matching `^tion$` to choose between mention and station). This approach guarantees that each guess fills an actual blank in a real word rather than merely splitting the pool of candidates. Because it only accepts letters that complete one of the surviving words, it never suggests irrelevant or low-probability letters. If somehow no match emerges, you fall back to your overall letter frequency ranking so the solver always has a valid move.

Results:

- 1,500 consecutive practice games
- 81.5 % overall win rate
- 1.7 wrong guesses per game on average