# Hangman Solver Agent

**BY: GURANSH SINGH**

## Introduction:-

This report documents the iterative process undertaken to develop an effective automated Hangman player. The goal was to maximize the win rate against a challenging set of words provided via an API and the local testing set. The development started with basic methods and progressed through advanced deep learning models, eventually leading to a hybrid strategy that leverages the strengths of different approaches based on the game's state.

*(As integrating N-gram analysis was forbidden, all approaches used Deep Learning based methods. Also all pretrained models were avoided as they were trained on data outside the training words provided.)*

**Dataset**: words_250000_train.txt (approx. 227k unique words) divided into train(205K), validation(20k) and test(2K) using stratified splitting to maintain the normal distribution across all the set and minimize KL-divergence affecting inference.

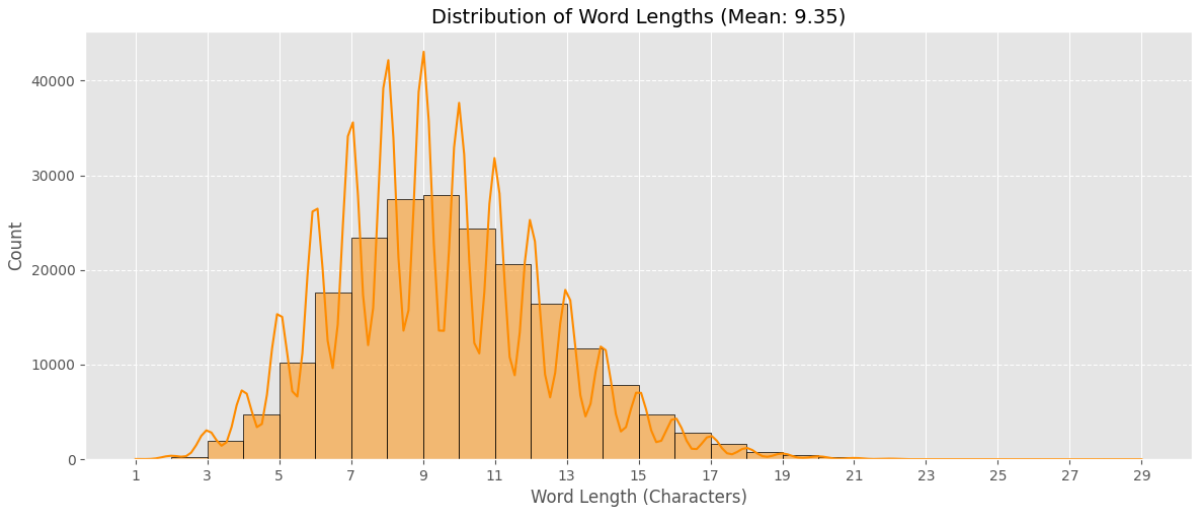


*Fig. Word length analysis of the dataset. It is also representative of the general distribution found in English language.*

# My Approach: -

1. Gated BiLSTM with Attention: -
    - A more sophisticated approach using a neural network was developed. The architecture consisted of:
        - An Embedding layer.
        - A multi-layer Bidirectional LSTM (BiLSTM) to capture sequential information.
        - A Multi-Head Self-Attention mechanism to weigh important characters.
        - A Gating mechanism (using an MLP) to dynamically fuse the sequence representation (O_Attn) with a projected representation of the game state (C_State, a 52D vector encoding visible letters and incorrect guesses). After trying various fusion mechanism, the gated fusion in the script proved to be the best to provide the necessary context while balancing the noise of the data that represent two different context entirely.
        - A final fully connected layer to predict logits for each character in the vocabulary.
    - **Training**: This model was trained on the expert_train.txt dataset, random simulating game states and predicting the next letter.
    - **Performance**: Local evaluation on test set achieved approximately 62-63% game win accuracy. This model was integrated into the HangmanAPI script, replacing the basic frequency guesser.

**2. Frequency Fusion: Combining LSTM and Dictionary Frequencies**

- The LSTM model excels at pattern recognition and context but doesn't inherently know if a predicted word is valid. The dictionary frequency (implemented via filtering/counting) method is grounded in valid words but lacks the LSTM's contextual understanding. Combining them could leverage both strengths.
    1. Get LSTM probabilities for unguessed letters (lstm_probs).
    2. Filter the dictionary based on the current pattern and incorrect guesses (valid_words).
    3. Calculate softmax probabilities based on letter frequencies within valid_words (frequency_probs).

4. Fuse the scores using a weighted average: final_score = (1 - alpha) * lstm_p + alpha * frequency_p.

- **Alpha Tuning:** Experiments showed alpha = 0.35 yielded the best results (highest Net Guess Impact).
- **Performance:** This fusion improved local accuracy by **1-2%**.

| Metric | Value | Percentage |
|---|---|---|
| Total Guesses Analyzed | 20,754 | |
| Guesses where Fusion Changed Choice | 6,203 | 29.89% (of Total) |
| Beneficial (Changed to Correct) | 1,578 | 25.44% (of Changed) |
| Harmful (Changed to Wrong) | 1,279 | 20.62% (of Changed) |
| Neutral (No change in Correctness) | 3,346 | 53.94% (of Changed) |
| **Net Guess Impact (Beneficial-Harmful)** | **+299** | |

*Table. Impact of frequency fusion*

### 3. Adaptive Alpha

- Detailed analysis of the Frequency fusion revealed it performed poorly when only **one blank** remained (Net Impact: -7). In this specific scenario, the LSTM (trained to fill blanks) was a better specialist.
- The fusion logic was modified:
    - If num_blanks == 1, set current_alpha = 0.0 (use only LSTM).
    - Otherwise, use current_alpha = base_alpha (0.35).
- **Performance:** This corrected the negative impact for 1-blank cases and slightly improved overall local accuracy to another **1%** with a higher Net Impact.

| Blanks Remaining | Total Guesses | Decision Rate | Net Impact |
|---|---|---|---|
| 1 Blank Left | 1,936 | 1.60% | **-7** |
| 2-3 Blanks Left | 5,185 | 15.58% | +44 |
| 4-5 Blanks Left | 5,016 | 26.36% | +24 |
| 6+ Blanks Left | 8,617 | 46.91% | +238 |

*Table. Blank( masked positions ) affect, before adaptive flow.*

### 4. Short Word Specialization

- This addresses the persistent low accuracy on short words.
- **Fine-tuned** the high-performing base model using *only* the short words (length <= 5) from the training/validation sets, using a low learning rate.
- **Discovery:** Testing revealed that this fine-tuned short model performed *best on its own* (alpha=0.0, no Frequency fusion) for short words, achieving **+3% win rate** on the short test set, outperforming the base model.

| Word Length | Total Guesses | Accuracy | Decision Rate | Net Impact |
|---|---|---|---|---|
| Short (<=5) | 1,365 | 31.43% | 43.88% | +26 |
| Medium (6-8) | 6,476 | 51.20% | 34.73% | +110 |
| Long (9-12) | 9,462 | 68.98% | 28.32% | +130 |
| Very Long (13+) | 3,465 | 84.33% | 20.61% | +21 |

*Table. Word length Accuracy analysis.*

### 5. Hybrid Strategy (Final Approach)

- Combining the best strategies identified for different word lengths.
- **Final Flow:-**

    - **If word_length <= 5:** Use the **Fine-Tuned Short Model** directly (no Frequency fusion, equivalent to alpha = 0.0).
    - **If word_length > 5:** Use the **Base Model** combined with the **Adaptive Frequency Fusion** (alpha = 0.0 if 1 blank left, otherwise alpha = 0.35).
- **Performance:** This hybrid approach yielded the best overall local accuracy on the full test set: **64.15%**.

## Other Approaches Tried:-

1. **Transformers:**
    - **Result:** ~51% (Underperformed 62% BiLSTM baseline).
    - **Conclusion:** Less effective than BiLSTM for this task/data, as it is known for requiring larger amounts of data and pre-training on much larger token counts.
2. **Reinforcement Learning (RL):**
    - **End-to-End Model (PPO / DQN):**
        - **Result:** PPO: ~13.5%; DQN: Very poor results (failed to converge or were unstable).
        - **Conclusion:** Catastrophic forgetting occurred (PPO). DQN unstable for this state/action space.
    - **PPO Meta-Agent (Re-ranker):**
        - **Result:** ~+0.9% gain (on LSTM+frequency expert), +0.0% gain (on LSTM-only expert).
        - **Conclusion:** Marginal benefit only when arbitrating between LSTM & frequency. Limited by the frozen expert and the reward function learnt to greedily trust the underlying base model for high rewards. The rewards increase smoothly from -ve to high +ve counts but provided not benefits. This could be clearly explained as
    - **Monte-Carlo Tree Search:**
        - Standard MCTS not directly applicable. Hangman is an **imperfect information** game (secret word hidden). Standard MCTS requires perfect information. When tried MCTS with our base model guiding the search, it achieved 98% win rate( Oracle approach where the search could check if its correct and then backpropagate). As in real

game play against the server, the word is not revealed until the choice is made.

3. **Gradient Boosting (CatBoost/XGB Stacking):**
    - **Result:** ~49% validation accuracy (next letter) and 58%( regression re-ranker)
    - **Conclusion:** objective mismatch, slow data generation. Didn't clearly beat the fusion approach. Also the gains noticed with increase in data were relatively minimal.

## Future Directions :-

1. **Length-Specific Model Ensemble:** The hybrid strategy demonstrated the benefit of a specialized model for short words (<=5). This could be extended by training separate models (or fine-tuning the base model) for different word length bins (e.g., 6-8, 9-12, 13+). An orchestrator could then select the most appropriate model based on the current word's length, potentially improving accuracy within each specific range. (Potential gains: +5-10%)

2. **Expanded Training Corpus:** The current models were trained exclusively on the provided words_250000_train.txt. Training or pre-training on a significantly larger and more diverse corpus of English words could enhance the model's understanding of letter patterns and potentially improve performance, especially on less common words encountered during gameplay or those absent from the dictionary.

3. **N-gram Integration:** While N-gram analysis was explicitly excluded in this project, traditional N-gram models are known baselines for Hangman. If allowed, incorporating N-gram statistics (e.g., character transition probabilities) as an additional feature or score to be fused with the LSTM and frequency predictions could provide complementary information and potentially boost accuracy further. (Potential gains: +5-7%)

## Conclusion:-

The final Hangman agent uses a hybrid strategy refined through data analysis and trying many different approaches with the time , resource constraints and Hangman being an imperfect information game(secret word hidden) . A base BiLSTM+Attention model was improved by state and adaptive dictionary frequency fusion. Recognizing poor performance on short words, I fine-tuned a specialist model used *without* fusion for these cases. This hybrid approach—switching between the short-word specialist and the adaptively fused base model—achieved 64.15% locally, 65% on the API practice runs and 63.2% on the final 1000 recorded runs, outperforming Transformers, RL, and other methods explored for this problem.