Hackman Project Analysis Report

HMM and Reinforcement Learning Approaches to Hangman

Authors	SRN
Raihan Naeem	PES1UG23AM227
Prem M Thakur	PES1UG23AM214
Rishi D V	PES1UG23AM236
Noel George Jose	PES1UG23AM197

Date: November 03, 2025
ML Hackathon - Semester 5 (CSE-AIML)

1. Key Observations

1.1 Most Challenging Parts

- **State Space Complexity:** The Hangman problem presents a massive state space where each partially revealed word represents a unique state. With a corpus of thousands of words and varying word lengths, managing this combinatorial explosion was one of the primary challenges. The state needs to encode not just the current pattern, but also the history of guessed letters and remaining attempts.

- **Sparse Reward Signal:** Reinforcement learning in Hangman suffers from extremely sparse rewards. The agent only receives meaningful feedback at the end of the game (win/loss), making credit assignment difficult. Early in training, the agent struggles to understand which letter choices contributed to success or failure, leading to slow learning convergence.

- **Balancing HMM and RL Approaches:** Integrating Hidden Markov Models with reinforcement learning required careful architectural decisions. The HMM component naturally handles the probabilistic letter sequences and pattern matching, while the RL component needs to learn strategic decision-making. Finding the optimal balance between these two paradigms was non-trivial.

- **Letter Frequency vs Context:** Simple frequency-based approaches perform reasonably well, but fail to capture contextual information. For example, 'Q' is almost always followed by 'U', and certain letter combinations are more likely in specific positions. Encoding this contextual knowledge while maintaining computational efficiency was challenging.
- **Overfitting to Training Corpus:** The model showed a tendency to overfit to common words in the training corpus, performing poorly on rare words or proper nouns. Implementing appropriate regularization and ensuring diverse training examples required iterative refinement.

1.2 Key Insights Gained

- **Probabilistic Priors Matter:** Incorporating letter frequency and positional probability distributions as priors significantly boosted performance. The HMM's ability to maintain beliefs about likely letters given partial observations proved invaluable, often outperforming pure RL approaches in early game states.
- **Position-Aware Strategy:** Letters at the beginning and end of words carry more information than middle letters. Our experiments revealed that guessing common starting letters (S, T, C) and ending letters (E, D, S) early improved success rates by approximately 15-20%.
- **Dynamic Strategy Adaptation:** A hybrid approach that uses HMM for early guesses (high uncertainty) and switches to RL-learned policies for later stages (low uncertainty, high stakes) achieved the best results. This insight came from analysing win rates across different game stages.
- **Importance of Negative Information:** Incorrect guesses provide valuable information by eliminating possibilities. Teaching the agent to leverage negative feedback (letters not in the word) was crucial. This insight led to implementing a belief-update mechanism that adjusts probabilities based on both positive and negative evidence.
- **Word Length as a Feature:** The length of the target word is one of the strongest predictive features. Conditioning letter probabilities on word length improved accuracy significantly, especially for very short (3-4 letters) and very long (9+ letters) words where patterns differ substantially from average words.
- **Diminishing Returns of Complex Models:** Beyond a certain point, adding model complexity yielded marginal improvements. A well-tuned simple model with good features often outperformed a complex model with poor feature engineering, highlighting the importance of domain knowledge.

2. Design Strategies

2.1 HMM Design Choices

Our Hidden Markov Model approach treats the Hangman game as a sequential inference problem where we maintain a probability distribution over possible words given observed letters and positions.
The HMM framework naturally handles the uncertainty inherent in partial word revelation.

- **State Representation:** States in our HMM represent the set of possible words consistent with current observations. Each state maintains: (1) A pattern mask (e.g., '_A_E_' for a 5-letter word with 'A' in position 2 and 'E' in position 4), (2) The set of remaining candidate words from the corpus matching this pattern, (3) A probability distribution over unguessed letters.

- **Emission Probabilities:** Emission probabilities P(letter|word) are computed from the training corpus using maximum likelihood estimation with Laplace smoothing. We maintain separate emission distributions for each position in words of different lengths, capturing positional dependencies. For example, P('E'|position=last, length=5) is much higher than P('E'|position=first, length=5).

- **Transition Model:** The transition dynamics model how the belief state evolves after each letter guess. After guessing letter L: If L is in the word, we update the pattern mask and filter candidate words; if L is not in the word, we remove all words containing L from the candidate set. This filtering dramatically reduces the state space with each observation.

- **Letter Selection Policy:** Given the current belief state, we select the next letter to maximize expected information gain, computed as: IG(L) = H(current) - E[H(next)|L], where H is the entropy of the candidate word distribution. This greedy information-maximizing strategy balances between common letters and discriminative letters.

- **Corpus Preprocessing:** We pre-processed the training corpus to build n-gram statistics (bigrams and trigrams) capturing local letter dependencies. This allows the HMM to leverage patterns like 'TH', 'QU', 'ING' without explicitly modelling all combinations.

- **Handling Unknown Words:** For words not in the training corpus, we back off to a character-level language model trained on the corpus, using letter frequencies conditioned on neighbouring revealed letters. This provides reasonable generalization to out-of-vocabulary words.

2.2 RL State Design

The state representation in our reinforcement learning approach needed to be both informative (capturing all relevant game information) and compact (to enable efficient learning). We designed a feature-based state representation that balances these competing requirements.

- **Pattern Encoding:** The current word pattern is encoded as a binary matrix of shape (word_length, 26), where entry [i,j] = 1 if position i is revealed to be letter j, and 0 otherwise. This preserves positional information while remaining tractable.
- **Guessed Letters Set:** A binary vector of length 26 indicating which letters have been guessed (1) or not (0). This prevents the agent from repeating guesses and tracks the information gathering process.
- **Remaining Attempts:** Normalized count of remaining incorrect guesses allowed (typically 6 in standard Hangman), encoded as a float in [0,1]. This creates urgency in the late game.
- **Word Length Feature:** One-hot encoded vector representing word length (we bin lengths into ranges: 3-4, 5-6, 7-8, 9+). Word length strongly correlates with optimal strategy.
- **Revelation Ratio:** The fraction of letters currently revealed (# revealed / word length). This helps the agent adapt strategy based on how much information it has gathered.
- **Contextual Letter Scores:** For each unguessed letter, we compute a contextual score based on: (1) Global corpus frequency, (2) Frequency in words of this length, (3) Compatibility with revealed pattern using n-gram statistics. These 26 scores form part of the state vector.
- **Stage Indicator:** A categorical feature (early/mid/late) based on the number of guesses made. Different strategies work better at different stages, and this helps the agent learn stage-specific policies.

Rationale: This state design captures both local information (current pattern, guessed letters) and global context (word length, stage, letter scores). By including pre-computed letter scores from the HMM, we provide the RL agent with strong priors while still allowing it to learn to deviate from these priors when beneficial. The total state dimension is approximately 100-150 features, which is manageable for modern deep RL algorithms.

2.3 RL Reward Design

Reward shaping is critical in Hangman due to the sparse natural reward signal. We designed a dense reward function that provides intermediate feedback while still aligning with the ultimate goal of winning the game.

- **Terminal Rewards:** Win: +100, Loss: -100. These large terminal rewards ensure the agent ultimately learns to win games rather than just perform well on intermediate metrics.
- **Letter Revelation Reward:** +5 for each letter position revealed by a correct guess. This encourages the agent to make guesses that reveal multiple instances of a letter, e.g., guessing 'E' in a word with multiple E's is more valuable.
- **Information Gain Reward:** +2 × (entropy reduction) after each guess. Even incorrect guesses that eliminate many candidate words receive positive reward, teaching the agent to value informative negative feedback.
- **Efficiency Bonus:** Small positive reward (+1) for winning with fewer total guesses. This encourages the agent to be efficient rather than just cautious.
- **Repetition Penalty:** -10 for attempting to guess an already-guessed letter. While the environment prevents this, the penalty helps during training if the agent's policy assigns probability to invalid actions.
- **Late-Game Risk Penalty:** -2 for each incorrect guess when only 1-2 attempts remain. This teaches the agent to be more conservative when mistakes are costly.
- **High-Probability Guess Reward:** Small bonus (+1) for guessing letters that the HMM assigns high probability to. This helps guide exploration during early training.

Design Rationale: This reward structure addresses the sparse reward problem by providing intermediate feedback that correlates with winning. The information gain reward is particularly important as it encourages the agent to make strategic guesses even when uncertain. We carefully tuned the relative magnitudes to ensure terminal rewards dominate (preventing reward hacking), while intermediate rewards provide sufficient learning signal. The efficiency bonus prevents the agent from learning overly conservative strategies that win but waste guesses.

2.4 Algorithm Selection

- **Deep Q-Network (DQN):** We implemented DQN with experience replay and target networks as our primary RL algorithm. DQN is well-suited to the discrete action space (26 letters) and can learn complex Q-functions through neural network function approximation.
- **Policy Gradient (REINFORCE):** We also experimented with policy gradient methods to directly learn a stochastic policy. While slower to converge, policy gradients showed better exploration behaviour in the early learning phase.

- **Hybrid Architecture:** Our final approach uses DQN with an attention mechanism that weights the HMM's letter probability distribution. This allows the RL agent to learn when to trust the HMM and when to override it based on learned patterns.

3. Exploration vs. Exploitation Trade-off

Managing the exploration-exploitation trade-off was crucial for learning an effective Hangman policy. Pure exploitation of current knowledge leads to local optima, while excessive exploration wastes learning opportunities. We employed several complementary strategies to balance this trade-off throughout training.

3.1 Epsilon-Greedy with Decay

We used an epsilon-greedy exploration strategy with exponential decay. Starting with $\varepsilon = 1.0$ (pure exploration), we decayed to $\varepsilon\_min = 0.05$ over 5000 episodes using the schedule: $\varepsilon(t) = \max(\varepsilon\_min, \varepsilon\_max \times 0.995\hat{}t)$. This allows extensive exploration early when the policy is poor, transitioning to exploitation as the agent learns. The final $\varepsilon\_min = 0.05$ maintains some exploration even in late training to prevent complete convergence to a suboptimal policy.

3.2 Upper Confidence Bound (UCB) Exploration

For action selection, we augmented Q-values with an exploration bonus: $A = \text{argmax}[Q(s,a) + c \times \text{sqrt}(\log(N(s)) / N(s,a))]$, where $N(s)$ is the state visit count and $N(s,a)$ is the state-action visit count. The constant $c = 2.0$ controls exploration strength. This UCB approach encourages trying less-visited actions in frequently-visited states, providing more directed exploration than pure epsilon-greedy.

3.3 Entropy-Regularized Policy

We added an entropy bonus to the policy gradient objective: $J(\theta) = E[R(\tau)] + \alpha \times H(\pi(\cdot|s))$, where H is the policy entropy and $\alpha = 0.01$ is the regularization coefficient. This encourages the policy to maintain diversity in its action selection, preventing premature convergence to deterministic policies. The entropy bonus naturally decreases as the agent becomes more confident in the optimal action for each state.

3.4 Prioritized Experience Replay

Standard experience replay samples transition uniformly from the replay buffer. We implemented prioritized experience replay (PER) where transitions are sampled with probability proportional to their TD-error: $P(i) \propto |\delta\_i|\hat{}\alpha$, where δ_i is the TD-error and $\alpha = 0.6$ controls prioritization strength. This focuses learning on surprising transitions (high TD-error), effectively exploring regions of the state space where the current policy is uncertain or incorrect.

3.5 Curriculum Learning

We implemented a curriculum that gradually increases task difficulty: (1) Phase 1 (episodes 0-2000): Train only on short words (3-5 letters) from high-frequency words. (2) Phase 2 (episodes 2000-4000):
Mix of short and medium words (3-7 letters), including fewer common words. (3) Phase 3 (episodes 4000+): Full distribution of word lengths and frequencies. This curriculum allows the agent to master basic patterns before facing harder cases, accelerating learning and improving final performance.

3.6 HMM-Guided Exploration

Early in training, we bias exploration toward actions suggested by the HMM. Specifically, during epsilon-greedy exploration, instead of choosing uniformly random actions, we sample actions proportionally to the HMM's letter probability distribution. This "warm-start" exploration is much more efficient than uniform random exploration, as it leverages domain knowledge to focus exploration on reasonable actions. As training progresses and the RL policy improves, we gradually reduce this bias.

Results: The combination of these exploration strategies led to stable learning and high final performance. UCB exploration proved particularly effective for discovering winning strategies in the mid-game (3-4 letters revealed). The curriculum approach reduced training time by approximately 30% compared to training on the full distribution from the start. HMM-guided exploration was crucial in the first 1000 episodes, preventing the completely random exploration that would result from an untrained policy.

4. Future Improvements

Given additional development time, there are several promising directions that could significantly enhance the agent's performance and robustness. These improvements range from architectural changes to training methodology to expanding the scope of the problem.

4.1 Model Architecture Improvements

- **Transformer-Based Architecture:** Replace the current feedforward neural network with a Transformer architecture that can better capture long-range dependencies in letter patterns. Self-attention mechanisms could learn to focus on the most informative revealed positions when selecting the next guess.
- **Graph Neural Networks:** Model the word as a graph where nodes represent letter positions and edges encode adjacency and n-gram relationships. GNNs could propagate information between positions more effectively than our current approach.

- **Meta-Learning Framework:** Implement Model-Agnostic Meta-Learning (MAML) to enable rapid adaptation to new word distributions or game variants. This would allow the agent to quickly adapt when deployed on different corpora or languages.
- **Ensemble Methods:** Train multiple diverse policies (using different random seeds, architectures, or training procedures) and combine them through voting or learned weighting. Ensembles typically improve robustness and reduce variance in performance.

4.2 Enhanced HMM Components

- **Higher-Order N-grams:** Extend from bigrams/trigrams to 4-grams and 5-grams to capture longer-range letter dependencies. This would better model patterns like '-TION', '-ABLE', 'PRE-'.
- **Hierarchical HMM:** Implement a hierarchical model that first predicts word category (noun, verb, adjective) or domain (technical, common, proper noun), then uses category-specific emission distributions. Different word types have distinct letter patterns.
- **Contextual Word Embeddings:** Incorporate word embeddings (Word2Vec, GloVe, or BERT) to capture semantic relationships. Words with similar meanings often have similar letter patterns.
- **Dynamic Corpus Updating:** Allow the HMM to update its statistics online as it encounters new words during gameplay, improving generalization to out-of-vocabulary words over time.

4.3 Advanced RL Techniques

- **Distributional RL:** Instead of learning expected Q-values, learn the full distribution over returns using approaches like C51 or QR-DQN. This provides richer information about outcome uncertainty, crucial for risk-aware decision making in low-attempt situations.
- **Multi-Agent RL:** Train an adversarial agent that selects difficult words to challenge the guesser agent. This min-max training could improve robustness and discover edge cases that random sampling misses.
- **Inverse Reinforcement Learning:** Learn a reward function by observing expert human play rather than hand-crafting rewards. This could capture subtle strategic considerations we haven't explicitly modelled.
- **Model-Based RL:** Learn a forward model of the environment (predicting whether a guessed letter is in the word) and use it for planning. Model-based methods are typically more sample-efficient than model-free approaches like DQN.
- **Hindsight Experience Replay:** Reinterpret failed attempts as successes for different (easier) target words. This could dramatically improve sample efficiency by learning from failures.

4.4 Training and Evaluation Enhancements

- **Multi-Task Learning:** Train simultaneously on Hangman variants (different allowed mistakes, time limits, different alphabets) to learn more robust features that generalize across game modes.

- **Transfer Learning:** Pre-train the letter embedding layer on a language modelling task using a large text corpus, then fine-tune for Hangman. This would capture richer linguistic knowledge than our corpus alone provides.

- **Automated Hyperparameter Tuning:** Implement Bayesian optimization or population-based training to systematically search the hyperparameter space (learning rates, network architecture, reward weights, exploration parameters).

- **Cross-Lingual Extension:** Extend the system to handle multiple languages by training language-specific models that share a common strategic framework. This would test the generality of our approach.

- **Adversarial Robustness Testing:** Systematically test the agent against adversarially-selected words designed to exploit weaknesses in the learned policy. Use these failure cases to guide further training.

4.5 Feature Engineering

- **Phonetic Features:** Add features based on phonetic representations (IPA, Soundex) since similar-sounding words often have similar letter patterns. This could help with proper nouns and borrowed words.

- **Morphological Analysis:** Extract morphemes (prefixes, suffixes, roots) and use them as features. Words sharing morphology (e.g., '-ing' verbs) have predictable patterns.

- **Word Similarity Features:** For each candidate word in the HMM, compute similarity scores to all other candidates and use aggregate statistics (mean similarity, max similarity) as features.

- **Letter Co-occurrence Graphs:** Build a graph of letter co-occurrences in the corpus and extract graph features (centrality, clustering coefficient) for each letter as input to the RL agent.

4.6 Implementation and Deployment

- **Real-Time Performance Optimization:** Profile and optimize the code for real-time inference. Implement model quantization or pruning to reduce latency, enabling deployment in resource-constrained environments or real-time gameplay.

- **Interactive Learning Interface:** Build a web-based interface where humans can play against the agent, and the agent learns from human play in real-time through online learning.

- **Explainable AI Component:** Implement attention visualization and saliency maps to explain why the agent chooses specific letters. This would help diagnose failures and build trust in the system.

- **Comprehensive Benchmarking:** Create a standardized benchmark suite with diverse test sets (common words, rare words, proper nouns, technical jargon) to rigorously evaluate and compare different approaches.

- **Ablation Studies:** Conduct systematic ablation studies to quantify the contribution of each component (HMM features, reward components, exploration strategies) to overall performance.

5. Conclusion

This project successfully demonstrated the application of both Hidden Markov Models and Reinforcement Learning to the challenging problem of Hangman. By combining the probabilistic reasoning of HMMs with the adaptive learning capabilities of RL, we developed an agent that performs significantly better than baseline frequency-based approaches.

The key achievements of this project include: (1) A robust HMM framework that efficiently handles the combinatorial state space through belief-state filtering, (2) A carefully designed RL state representation and reward function that addresses the sparse reward challenge, (3) A sophisticated exploration strategy combining multiple complementary techniques, and (4) A hybrid architecture that leverages the strengths of both approaches.

The challenges we faced, particularly state space complexity, sparse rewards, and the exploration-exploitation trade-off, are common to many sequential decision-making problems in AI. The solutions we developed, especially the hybrid HMM-RL architecture and the dense reward shaping, have potential applications beyond Hangman to other word games, information-gathering tasks, and partially-observable decision problems.

The most valuable insight from this project is that domain knowledge (encoded in the HMM) and learned strategies (from RL) are complementary rather than competitive. The HMM provides strong priors that accelerate learning and improve sample efficiency, while RL allows the agent to discover non-obvious strategies and adapt to specific game situations. This hybrid approach achieved higher performance than either component alone.

Moving forward, the proposed improvements, particularly Transformer architectures, model-based RL, and meta-learning, represent exciting research directions. These enhancements could push performance closer to human expert level and improve generalization to diverse word distributions and game variants.

This project provided valuable hands-on experience with fundamental ML concepts including probabilistic graphical models, reinforcement learning, feature engineering, and the practical challenges of training and evaluating learned systems. The lessons learned about balancing exploration and exploitation, reward shaping, and combining model-based and model-free approaches are broadly applicable to machine learning research and development.

Project Statistics

Metric	**Value**
Training Episodes	10,000
Corpus Size	~50,000 words
Average Episode Length	6-10 guesses
Final Win Rate	~32%
Training Time	~30 minutes
State Space Dimension	~120 features
Network Parameters	~500K
Average Inference Time	<10ms