Hackman Project - Analysis Report

Project: Hangman AI using HMM and RL
Date: November 3, 2025
SRNs: PES2UG23CS401, PES2UG23CS407, PES2UG23CS412, PES2UG23CS416
Names (in order of SRN): Nitin Reddy, Pavan Saxena, Podili Sipivishta, Poorva Tejas Reddy

1. Key Observations

Most Challenging Parts

Challenge 1: Severe Overfitting - Training/Validation: 93.5% win rate - Test Performance: 8.3% win rate - 11x performance drop due to vocabulary mismatch - Training corpus had common words, test had rare/technical words - Pattern matching failed on unseen vocabulary

Challenge 2: State Space Explosion - 260,848 unique states after 50,000 episodes - Tabular Q-learning has zero transfer between states - Test words created entirely new states - Q-table size: ~54 MB

Challenge 3: Repeated Guess Prevention - Must check if letter already guessed BEFORE adding to set - Penalty: -10 reward without reducing lives - Required careful state management

Challenge 4: Reward Shaping - Started with sparse rewards (only win/loss) - slow learning - Evolved to proportional rewards: +5 per letter revealed - Final: +50 win bonus, -30 loss penalty, -10 repeated guess - Dense rewards accelerated learning significantly

Key Insights

1. **Memorization vs Generalization**: Tabular Q-learning memorizes patterns but cannot generalize to new vocabulary. No overlap between train/test states.
2. **Misnamed Algorithm**: Called it "HMM" but it's actually a corpus-filter with frequency counting, not a true Hidden Markov Model.
3. **Distribution Validation**: Validated on same corpus as training (93.5%) but test had different vocabulary (8.3%). Should validate on realistic test samples.
4. **Simple Beats Complex**: Position-frequency method (19.5%) outperformed Q-learning (8.3%) by 2.3x with less complexity.
5. **Caching Critical**: Without caching: 41 minutes training. With caching: 5 minutes (8x speedup). 90% cache hit rate.

---

2. Strategies

HMM (Corpus-Filter) Design

Pattern-Based Filtering

```
# Filter corpus to words matching pattern
matches = [w for w in corpus if matches_pattern(w, pattern)]
# Count letters only in blank positions
for word in matches:
    for i, ch in enumerate(word):
        if pattern[i] == '_':
            counts[ch] += 1
```

Key Decisions: - Filter by pattern before counting (high accuracy when vocabularies match) - Count only blank positions (ignore revealed letters) - Laplace smoothing (add 1e-6 to prevent division by zero) - Cache
with (pattern, guessed) key (8x speedup) - Normalize after zeroing out guessed letters

Trade-offs: - Pros: Accurate on known patterns, fast with caching - Cons: Fails on unseen patterns, no generalization

RL State Design

State Components:

```
state = (
masked_pattern,# "_a_e"
    frozenset(guessed_letters), # {'a', 'e'}
lives_remaining,# 5
    top_3_oracle_letters        # ('k', 'c', 't')
)
```

Rationale: - Pattern: Essential game state - Guessed: Prevents repeated guesses, hashable with frozenset - Lives: Affects risk strategy - Oracle top-3: Provides corpus knowledge without explosion **State Space:** - Theoretical: 10^16 states - Actual: 260,848 states (0.0000000026% sparsity)

RL Reward Design

Reward Structure:

```
if repeated:

    reward = -10
elif correct: reward = 5 * num_revealed #
    Proportional!
    if won:
        reward += 50
else:
    reward = -5 if
    lost: reward += -
    30
```

Design Principles: - Dense rewards every step (not just terminal) - Proportional to value (revealing [1]letters better than 1) - Win bonus (50) > Loss penalty (30) - Repeated guess heavily penalized (-10)

[1] . **Exploration vs Exploitation**

Epsilon-Greedy with Decay

Evolution: - V1 (Sparse): +1/-1 only at end, 30% win rate - V2 (Simple): +1/-1 every step, 75% win rate V3 (Proportional): +5 x reveals, 89.7% win rate

3. Exploration vs Exploitation

Epsilon-Greedy with Decay

Configuration:

```
EPSILON_START = 1.0       # 100% exploration
EPSILON_DECAY = 0.995     # Decay each episode
EPSILON_MIN = 0.05        # Minimum 5%
```

Decay Schedule: - Episode 0: 100% exploration - Episode 100: 60.6% exploration - Episode 500: 8.2% exploration - Episode 1000+: 5% exploration (minimum)

Intelligent Exploration

Key Innovation: Sample from Oracle distribution during exploration, not uniform random. `if`

```
random() < epsilon:
    # Oracle-weighted exploration probs =
    normalize(oracle_probs[available_actions]) return
    np.random.choice(available_actions, p=probs)
else:
    # Exploitation: Hybrid Oracle + Q-learning
    score = 0.75 * oracle_prob + 0.25 * q_value
    return argmax(score)
```

Benefits: - Explores promising actions more often - Better performance during training - Leverages domain knowledge - Faster learning than uniform exploration

Hybrid Weighting

Oracle vs Q-Learning Balance: - HMM_WEIGHT = 0.75 (75% Oracle, 25% Q-learning) - High Oracle weight helps training but causes overfitting - Should use lower weight (0.3-0.5) for better generalization

During Evaluation: - Set epsilon = 0.0 (pure exploitation) - No exploration during testing - Deterministic, repeatable results

---

4. Future Improvements

1. **Adaptive HMM Weighting** - Adjust weight based on pattern matches - Many matches: Trust Oracle (0.8) - Few matches: Trust Q-learning (0.2) - Expected: +4% improvement

2. **N-gram Letter Sequences** - Track bigrams like “qu”, “th”, “ing” - After ‘q’ revealed, strongly prefer ‘u’ Expected: +3% improvement

---

Configuration:

3. **Position-Frequency Fallback** - When no corpus matches, use position frequencies - graceful degradation on rare patterns - Expected: +6% improvement

Combined Impact: 8.3% to 21.3%

4. **Deep Q-Network (DQN)** - Replace tabular with neural network - Feature-based generalization

5. **Transfer Learning** - Pre-train on large dictionary (500k words) - Fine-tune on competition corpus - Interpolate based on confidence - Expected: +7% improvement

6. **Ensemble Methods** - Combine Q-learning, corpus-filter, position-freq, bigram - Weighted voting with position-freq as primary (40%) - Expected: +4.5% improvement

Combined Impact: 21.3% to 30%

7. Contextual Bandits - Thompson sampling for adaptive exploration 8. Seq2Seq Neural Model - LSTM encoder-decoder with attention 9. Meta-Learning - Fast adaptation to new vocabulary distributions Potential: 30-35% win rate

5. Summary

Main Takeaways

1. Overfitting is real: 93.5% validation to 8.3% test
2. Simple beats complex: Position-frequency (19.5%) > Q-learning (8.3%)
3. Distribution matters: Validate on realistic test data
4. Naming precision: Corpus-filter is not HMM
5. Reward shaping helps: Dense, proportional rewards
6. Intelligent exploration: Oracle-weighted > uniform random
7. Caching critical: 8x speedup

Performance Summary

Method	Win Rate	Score	Complexity
Q-Learning + Oracle	8.3%	-57,534	High
Pure Oracle	3.2%	-58,700	Medium
Position Frequency	19.5%	-56,010	Low

Training Results

- Episodes: 50,000
- Training Win Rate: 89.7%
- Validation Win Rate: 93.5%
- Test Win Rate: 8.3%
- Q-table States: 260,848
- Training Time: 5-10 minutes

Hyperparameters

- Learning Rate: 0.1
- Discount Factor: 0.95

- Epsilon: 1.0 to 0.05 (decay 0.995)
- HMM Weight: 0.75
- Max Lives: 6
