Learning-Based Decision Making in a Competitive Card Game

Master's Thesis
February 2026

Written by
Tamás Loós
Supervised by **Alvaro Torralba**

Computer Science
Department of Computer Science
Aalborg University

Dept. of Computer Science
Selma Lagerløfsvej 300
DK-9220 Aalborg
http://www.cs.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Learning-Based Decision Making in a Competitive Card Game

Theme:
Imitation Learning, Behavioral Cloning, Game AI, Neural Networks, Pointer Networks, DAgger, Card Games

Project Period:
Spring Semester 2026

Participant(s):
Tamás Loós

Supervisor(s):
Alvaro Torralba

Copies: 1

Date of Completion: February 24, 2026

Page Numbers: 46

Abstract:

This thesis investigates behavioral cloning for learning to play Scripts of Tribute, a competitive deck-building card game used as a testbed at the IEEE Conference on Games AI Competition. A neural network was trained to imitate an expert MCTS bot using 6,400 games (673,619 decisions), achieving approximately 59% top-1 accuracy, approximately 15% above the strongest trivial strategy. This accuracy plateau persisted across five model configurations (84K–2.3M parameters) and two training methods (behavioral cloning and DAgger), suggesting the bottleneck is not model capacity but the teacher's reliance on search information unavailable to the student. A pointer-based architecture addresses a mismatch between positional action encoding and input processing that ignores card order, scoring actions by card content rather than position. The deployed bot (457K parameters) makes decisions in approximately 5 ms and achieves 36.9% overall win rate against eleven competition bots, beating heuristic opponents but winning only 12-15% against the strongest MCTS-based bots. This gap is consistent with the compounding-error problem of behavioral cloning.

Thesis Summary

This thesis investigates whether a neural network can learn to play the competitive deck-building card game Scripts of Tribute by imitating an expert bot through behavioral cloning. Scripts of Tribute features hidden information, stochastic elements, and large branching factors, with a typical game involving approximately 12 turns per player and 9 decisions per turn (∼105 decisions total). The game has served as a testbed for AI research through the Scripts of Tribute AI Competition (TOTAIC) at the IEEE Conference on Games, where the strongest bots rely on Monte Carlo Tree Search (MCTS) with hand-crafted evaluation functions.

The expert teacher is SakkirinaSolo, the tournament-winning MCTS bot from TOTAIC 2025 (71.8% overall win rate). A dataset of 6,400 games (673,619 teacher decisions) was collected across four opponent types. The project follows a structured experimental progression investigating the initial system, data scaling, feature engineering, model architecture, distribution shift correction, and final evaluation.

The initial system (Experiments 0-2) established a two-layer MLP (1.77M parameters) trained on the game state encoded as a 2,840-dimensional feature vector. Starting at ∼56% top-1 validation accuracy on 800 games, regularization techniques (dropout, weight decay, batch normalization) were applied to control severe overfitting but did not meaningfully improve accuracy. Scaling the dataset from 800 to 6,400 games yielded a gain to ∼59%, with diminishing returns beyond 4,000 games. For context, trivial reference strategies achieve 29% (random legal) and 44% (most frequent legal action), so the model's accuracy is around 15 percentage points above the strongest trivial strategy. A legal-action-restricted weighted ROC-AUC score of 0.854 indicates that the model produces informative action rankings even when it does not select the teacher's exact first choice.

Feature engineering (Experiment 3, Section 5.1) highlighted a consequence of the positional action space design: action indices reference card positions, not card identities. Sorting cards within their zones lowered accuracy (51.6%, a 7.3 percentage drop) because the reordering broke the correspondence between state features and action labels. This motivated a redesign of the model architecture.

The resulting V2 architecture (Experiment 4, Section 5.2) replaced the fixed output layer with a pointer-based action head inspired by Pointer Networks [18]. Instead of mapping a trunk representation to 106 output logits via a linear layer, the model scores each candidate action by computing the dot product between a learned query vector and the corresponding card embedding. This content-based

scoring is independent of card order. The V2 architecture matched V1 accuracy ($\sim$59%) while decoupling action scoring from card positions. Scaling model size produced no further gains, confirming that model capacity is not the limiting factor.

The trained model was deployed as a live bot (LT_NN) via integration with the C# game engine. Despite $\sim$59% offline accuracy, LT_NN achieved only 12% win rate against the teacher. This gap is explained by compounding errors: each imperfect decision pushes the game into states not encountered during training, where subsequent predictions become less reliable. Three iterations of DAgger (Experiment 5, Section 6.2) added 169,000 teacher corrected samples from states visited by the learner, but produced no consistent improvement in either offline accuracy or live win rate. DAgger's formal convergence guarantees assume learners that can find globally optimal policies on the aggregated dataset [14], a condition that neural networks, with their non-convex loss landscapes, do not satisfy [8]. However, DAgger has been successfully applied with neural networks in other domains, so this theoretical gap alone does not fully explain the negative result.

Final evaluation (Chapter 7) assessed the deployed bot across multiple dimensions. Against eleven bots from the COG 2025 competition (500 games each), LT_NN achieved 36.9% overall win rate - consistently beating heuristic bots (99.8% vs Vei) but winning only 12-15% against the strongest MCTS opponents. Per-action-type analysis showed accuracy ranging from 92.6% (END_TURN) to 35.9% (CALL_PATRON), with patron-calling decisions being the primary weakness. Component ablation confirmed hand cards as the dominant input signal (53% relative accuracy drop when removed). The model makes decisions in approximately 5 ms.

The central finding is a persistent $\sim$59% accuracy plateau observed across five model configurations (84K to 2.3M parameters), two training methods (behavioral cloning and DAgger), and varying data volumes. This plateau is attributed to an information gap between teacher and student: the MCTS teacher accesses future states through search, evaluating long-term consequences of each move, while the student learns only from the current visible game state. Some teacher decisions are not predictable from the student's input features. The thesis contributes a pointer-based architecture that decouples action scoring from card positions, a complete pipeline from data collection to live deployment, evaluation of behavioral cloning's limits in this domain, and a documented negative result on DAgger.

Contents

1 Introduction

1.1 Context and Motivation

Deck-building card games present a challenging domain for artificial intelligence due to their combination of hidden information, stochastic elements, large branching factors, and long-term strategic planning. Scripts of Tribute (originally Tales of Tribute), a two-player deck-building card game introduced in The Elder Scrolls Online, has recently served as a testbed for game AI research through the Scripts of Tribute AI Competition (TOTAIC), held alongside the IEEE Conference on Games [7]. In the 2025 edition, the strongest bots relied on Monte Carlo Tree Search (MCTS) [3, 6] with handcrafted evaluation functions - an approach that requires substantial domain expertise and engineering effort. An alternative is to learn a policy directly from expert demonstrations using neural networks, an approach known as behavioral cloning or imitation learning [13].

1.2 Objective

This thesis investigates whether a neural network can learn to effectively play Scripts of Tribute by imitating an expert bot through behavioral cloning. The expert teacher is SakkirinaSolo, the tournament-winning MCTS bot from TOTAIC 2025 (71.8% win rate). The investigation is organized around four research questions:

1. **Can behavioral cloning learn effective play from expert demonstrations?** (Experiments 0-2, Chapter 4)
2. **Can improved representations improve accuracy further?** (Experiments 3-4, Chapter 5)
3. **Can distribution correction methods improve play?** (Experiment 5, Section 6.2)
4. **How does the final system perform against competition level bots?** (Chapter 7)

1.3 Contributions

The main contributions of this thesis are:

1. **A pointer-based neural network architecture** for action selection in card games, motivated by the observation that the initial positional action encoding is incompatible with input processing that ignores card order. The architecture scores each action by computing the dot product between a learned query and the corresponding card embedding, decoupling action selection from card position.
2. **A complete pipeline** from game log collection to live bot deployment, with systematic experimental investigation across six experiments.
3. **Evaluation** of the limits of behavioral cloning in this domain: a persistent $\sim$59% accuracy plateau, assessed through multiple methods including component ablation, per-action-type analysis, reference strategy comparisons, ranking quality metrics, and tournament evaluation against eleven competition bots.
4. **A documented negative result** on DAgger, with discussion of why the theoretical guarantees do not extend to neural network function approximators.

1.4 Code Availability

The source code for this thesis is available in two repositories:

- **Neural network code**: `https://github.com/Yhennon/master-nn` - data processing, model implementations, training scripts, experiment code, evaluation, and the deployed bot.
- **Game engine and infrastructure**: `https://github.com/Yhennon/MasterProject` - the C# game engine, game runner, DAgger wrapper, and tournament infrastructure.

2 | Background

This chapter introduces the game domain, the machine learning foundations used throughout the thesis, and the related architectural concepts that inform the model design.

2.1 Scripts of Tribute

Scripts of Tribute is a two-player deck-building card game. Each player begins with a small starter deck and acquires new cards from a shared tavern during play. The game features:

- **Patron system:** 4 patrons (chosen during a draft phase) + 1 Treasury patron, each providing unique abilities when called
- **Three win conditions:** accumulating 40+ prestige (opponent gets one response turn), reaching 80+ prestige (instant win), or gaining the favor of all patrons simultaneously
- **Hidden information:** opponent's hand and deck composition are not visible
- **Stochastic elements:** cards drawn from shuffled decks each turn and random tavern refreshes when purchase slots open up

A typical game lasts approximately 12 turns per player, with each turn involving multiple decisions - playing cards from hand, buying from the tavern, activating agents, calling patrons, and ending the turn. The teacher averages roughly 9 decisions per turn, totaling 105 individual decisions per game on average.

2.2 Neural Networks

A neural network is a parameterized function approximator that learns mappings from input data to output predictions. Neural networks can approximate complex nonlinear mappings from game states to actions directly from data, without requiring hand-crafted decision rules. The basic building block is a **layer**: a linear transformation of the input (multiplication by a weight matrix plus a bias vector), followed by a nonlinear **activation function**. Each output dimension of a layer is called a **neuron**. The activation function used throughout this thesis is ReLU (Rectified Linear Unit), which outputs $\max(0, x)$, passing positive values through unchanged and zeroing negative values.

A **multi-layer perceptron (MLP)** stacks multiple such layers in sequence. The V1 model in this thesis is a 2-layer MLP: the 2,840-dimensional input is linearly projected to 512 dimensions, passed through ReLU, projected to 512 dimensions again, through ReLU, and finally projected to 106 output dimensions (one per action). The final layer's outputs are called **logits**, which are raw scores that are converted into a probability distribution over actions via the **softmax** function, which exponentiates each logit and normalizes the results to sum to 1.

Several representational building blocks are used in the V2 architecture. **One-hot encoding** represents categorical variables (such as card deck type or card type) as binary vectors where exactly one element is 1 and all others are 0. **Embedding layers** are learned linear projections that map sparse one-hot vectors (mostly zeros) into shorter, dense vectors where each dimension is meaningful. For example, a 17-dimensional raw card encoding (concatenated one-hot vectors and scalars) is projected into a 64-dimensional learned representation. **Pooling** operations aggregate variable-length sets of embeddings into fixed-size summaries: **mean pooling** simply averages all embeddings, while **attention pooling** [9] computes a learned weighted average where the network decides which elements to weight more heavily. **Residual connections** [4] add the input of a block directly to its output (output $= x + f(x)$). Without residuals, stacking more layers can make training harder because the network must learn to pass information through unchanged, which is difficult for a chain of linear-then-nonlinear transformations. With residuals, a block can default to doing nothing ($f(x) = 0$ means output $= x$), so adding layers can only help. This stabilizes training.

Training proceeds by iterating over the dataset in **mini batches** (groups of 512 samples in this thesis). For each batch, the network performs a **forward pass** (computing predictions from inputs), evaluates a **loss function** that measures prediction error, and performs **backpropagation** to compute how each weight should change to reduce the loss. The weights are then updated by the optimizer. One complete pass through all training data is called an **epoch**. After each epoch, the model is evaluated on a held-out **validation set** (no weight updates) to monitor generalization. **Training loss** measures fit on the training data; **validation loss** measures performance on unseen data. When training loss continues decreasing but validation loss begins rising, the model is **overfitting** - memorizing training specific patterns rather than learning generalizable decision making.

Several **regularization** techniques are used to control overfitting, including early stopping, dropout [17], weight decay, batch normalization [5], and learning rate scheduling. These are introduced and evaluated in Experiment 1 (Section 4.4).

2.3 Behavioral Cloning

Behavioral cloning [13] learns a policy $\pi(a|s)$, a function that outputs the probability of choosing action a given game state s, by supervised learning on expert demonstrations. Given a dataset $\mathcal{D} = \{(s_i, a_i)\}$ of state-action pairs from an expert, the policy is trained to minimize the cross-entropy loss:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \log \pi(a_i|s_i)$$

Cross-entropy measures how well the model's predicted probability distribution matches the teacher's actual choices. It penalizes confident wrong predictions heavily: assigning 90% probability to the correct action restults in a low loss ($-\log 0.9 \approx 0.11$), while assigning only 1% results in a very high loss ($-\log 0.01 \approx 4.6$). This directly maximizes the probability assigned to the teacher's chosen action at each decision point.

A well-known limitation is **distribution shift** [14]: the learned policy encounters states during deployment that differ from the training distribution (which contains only states visited by the expert). Errors at these unfamiliar states compound over time, leading to accumulating failures.

2.4 DAgger

Dataset Aggregation (DAgger) [14] addresses distribution shift by iteratively:

1. Deploying the current learned policy to play games, collecting the sequence of states it visits
2. Having the expert label each of these learner visited states with the action it would have chosen
3. Aggregating the new labeled data with the existing dataset and retraining

This exposes the model to states it actually encounters during deployment, rather than only states the expert visits.

2.5 Pointer Networks

Pointer Networks [18] produce output sequences by pointing to positions in the input rather than projecting to a fixed output vocabulary. The core idea is to compute a compatibility score between a learned query (derived from the network's internal state) and each input element, then select the input with the highest score. This approach applies when the output space is defined by the input, for example, selecting which card to play from the current hand, where the set of candidates changes at every decision. The V2 architecture in this thesis borrows this scoring idea in a simplified form: a dot product between a query vector and each card's embedding (Section 5.2), without the full sequence-to-sequence attention mechanism of the original Pointer Networks.

3 | Methodology

This chapter describes the data collection pipeline, the action space design, the evaluation methodology, and the training setup shared across all experiments.

3.1 Data Collection

The teacher agent is SakkirinaSolo, an MCTS-based bot that won the TOTAIC 2025 tournament with a 71.8% overall win rate. A total of 12,800 games were collected across four opponent types (3,200 per matchup) using the Scripts of Tribute C# game engine. A custom logging wrapper was implemented to output JSONL files containing the full game state and the teacher's chosen action at each decision point.

The full 12,800-game dataset exceeded practical compute constraints for this project, so a smaller subset of 6,400 games (1,600 per matchup) was selected for training. The learning curve analysis in Section 4.5 confirms this is sufficient: accuracy at 4,000 games (58.83%) is nearly identical to 6,400 games (58.87%).

Matchup (Teacher vs)	Games	Decisions
MaxPrestigeBot	1,600	165,537
MCTSBot	1,600	183,238
RandomBot	1,600	144,304
SakkirinaSolo (mirror)	1,600	180,540
Total	**6,400**	**673,619**

Table 3.1: Dataset composition (training subset).

Only the teacher's decisions with a valid chosen action index are used. In the data collection setup, the teacher always plays as Player 1 and the opponent as Player 2. The opponent's decisions are not relevant since the goal is to imitate the teacher. Multi selection choices (where the teacher must select multiple items simultaneously) are excluded, affecting fewer than 0.25% of decisions.

3.2 Action Space

The action space consists of 106 discrete actions arranged in a fixed positional layout across six zones:

Zone	Indices	Slots	Limit Source
PLAY_CARD (hand)	0–16	17	Empirical (max observed: 5, padded for card-draw chains)
BUY_CARD (tavern)	17–21	5	Game rule (tavern always has exactly 5 cards)
CALL_PATRON	22–26	5	Game rule (4 chosen + 1 Treasury)
ACTIVATE_AGENT (own)	27–33	7	Game rule (max 7 agents per player)
ATTACK (enemy agents)	34–40	7	Game rule (max 7 agents per player)
CHOICE (card/effect)	41–104	64	Empirical (max observed: 21, padded 3x)
END_TURN	105	1	Always available
Total	**0–105**	**106**	

Table 3.2: Action space layout.

The action space is **positional**: action index 0 means "play the card in hand slot 0," not "play card X." The C# game engine presents actions as string commands (e.g., `"PLAY_CARD GOLD"`), and a custom mapping layer translates between these strings and the fixed 106-slot layout. Because action indices reference card positions, the state encoding must preserve positional information within each zone. Any transformation that reorders cards (like sorting) breaks the correspondence between state features and action labels, as demonstrated in Experiment 3 (Section 5.1).

At any given decision, typically only 2 to 15 of the 106 actions are legal. **Legal action masking** sets the logits of illegal actions to -10^9, ensuring illegal actions receive effectively zero probability and the model only predicts among legal alternatives. The fixed 106-slot layout simplifies training: mini-batch processing requires a fixed output size, and the cross-entropy loss works directly on the 106-dimensional logit vector.

3.3 Evaluation Methodology

The dataset is split into training (80%), validation (10%), and test (10%) sets at the **game level**, all decisions from a given game belong to exactly one split. This prevents data leakage, since consecutive decisions within the same game share most of their state. The split is stratified by matchup type to ensure each subset contains a balanced representation of all four opponent pairings. The test set is held out until final evaluation (Chapter 7).

Offline metrics:

- **Top-1 accuracy:** The fraction of decisions where the model's highest ranked prediction matches the teacher's exact choice.

- **Top-3 accuracy:** The fraction of decisions where the teacher's choice appears among the model's three highest ranked predictions.
- **Baseline comparisons:** To contextualize accuracy, two trivial baselines are used throughout. The **random-legal** baseline selects uniformly at random among legal actions, achieving approximately 29% accuracy. The **most-frequent-legal** baseline always selects the legal action with the highest frequency in the training set (in practice, always PLAY_CARD slot 0, the single most common action at 27.6% of decisions), achieving approximately 44% accuracy. These baselines establish the floor: the model's contribution is measured by its lift above the strongest trivial strategy.
- **ROC-AUC (Ranking Quality):** Top-1 accuracy does not capture how well the model ranks actions it does not select as the top choice. The ROC-AUC metric (Receiver Operating Characteristic - Area Under the Curve) measures ranking quality: for a given action, it answers whether the model assigns higher probability to samples where the teacher chose that action versus samples where the teacher chose something else. For multiclass problems, One-vs-Rest AUC is computed for each action and then averaged. An important adjustment is required: naive AUC computation includes samples where the action was not even legal, and since legal action masking assigns near zero probability to illegal actions, these easy negatives make the score look better than it really is. To avoid this, AUC for each action is computed using only samples where that action was a legal option. This variant is referred to as **legal-restricted** ROC-AUC throughout this thesis. This asks the meaningful question: when an action was available, did the model rank it higher when it was the correct choice? Detailed ROC-AUC results are presented in Section 7.3.

Live evaluation:

- **Win rate** against opponent bots (100–500 games per matchup). Scripts of Tribute has a first-player advantage (Player 1 moves first), so all win rate evaluations split games evenly between sides (e.g., 50 as Player 1 and 50 as Player 2) to avoid positional bias.

3.4 Training Setup

All experiments use the following unless otherwise noted. The optimizer is AdamW [10], a variant of stochastic gradient descent that adapts the learning rate separately for each weight based on the history of past gradients, and applies weight decay (L2 regularization) directly to the weights rather than through the gradient. This generally converges faster and more reliably than basic gradient descent. The best epoch reported in results tables is the epoch with the lowest validation loss.

Setting	Value
Hardware	NVIDIA RTX 2060 (6 GB VRAM), 16 GB RAM
Framework	PyTorch
Optimizer	AdamW (lr=10^{-3}, weight decay=10^{-4})
Batch size	512
LR scheduling	ReduceLROnPlateau (factor=0.5, patience=5)
Gradient clipping	Max norm 1.0 (limits gradient size to prevent unstable weight updates)
Early stopping	Patience 7 on validation loss
Seeds	42 (data split), 0 (model initialization)

Table 3.3: Training configuration.

4 | Baseline System

This chapter establishes the initial system: a flat state encoding, a simple MLP policy network, and two experiments studying the effects of regularization and dataset size on accuracy.

4.1 State Encoding (V1)

The V1 encoder converts the JSON game state into a fixed-length feature vector of 2,840 dimensions, structured as follows:

Zone	Slots	Dims/slot	Total dims
Global	1	15	15
Patrons	1	5	5
Hand	17	17	289
Tavern	5	17	85
Own agents	7	19	133
Enemy agents	7	19	133
Choice global	1	4	4
Choice slots	64	34	2176
Total			**2840**

Table 4.1: V1 feature vector layout.

Global features (15 dims) capture numeric game state: each player's power, coins, and prestige (6 total), plus pile and zone sizes - hand size, draw pile, cooldown pile, agent count, and tavern count for the current player, as well as hand size, draw size, cooldown size, and agent count for the opponent.

Patron features (5 dims) encode the favor state of each patron (including Treasury) as a single scalar: $+1$ if the patron favors the current player, -1 if the opponent, and 0 if neutral.

Each **card** (Hand and Tavern cards) is encoded as a 17-dimensional vector: 3 scalar features (cost, HP, taunt) + an 8-dimensional one-hot vector for deck type + a 6-dimensional one-hot vector for card type.

Agent slots extend this with 2 additional features (current HP, activated flag) for a 19-dimensional encoding per agent.

Choice global (4 dims) indicates whether a pending choice exists, its type (card or effect), and the number of available options. **Choice slots** use a 34-dimensional encoding: 3 binary flags (is_card, is_effect, is_skip) indicating the slot type, the 17-dimensional card encoding, and a 14-dimensional one-hot vector for effect type. For card choices, the card features are filled and the effect vector is zero; for effect choices, the reverse. The is_skip flag marks the optional "pass" slot when the choice allows zero selections.

The slot counts in this table mirror the 106-action output layout (Section 3.2): the 17 hand slots correspond to the 17 "play card" actions, the 5 tavern slots to the 5 "buy card" actions, and so on. Two limitations of this encoding are relevant for later experiments: (1) because of this positional correspondence, the model treats cards differently depending on which slot they occupy, even though the order within a hand or tavern has no gameplay significance; and (2) the 64 choice slots are mostly zeros, since the maximum observed number of pending choice options is 21.

4.2 Model Architecture (V1)

The V1 model is a 2-layer MLP with legal action masking:

$$\text{Input}(2840) \to \text{Linear}(512) \to \text{ReLU} \to \text{Linear}(512) \to \text{ReLU} \to \text{Linear}(106)$$

Each Linear layer performs a matrix multiplication plus bias: $\text{Linear}(x) = Wx + b$, where W is a learned weight matrix and b is a learned bias vector. The first layer transforms the 2,840-dimensional input into a 512-dimensional hidden representation, which is a compressed internal summary of the game state. The ReLU activation ($\max(0, x)$) introduces nonlinearity, enabling the network to learn relationships that a single linear transformation cannot capture (for example, that a card's value depends on what other cards are already in play). The second layer further refines this 512-dimensional representation through the same transformation and activation process. The final layer projects from 512 to 106 dimensions, producing one raw score (logit) per possible action.

Legal action masking. At each decision point, only a subset of the 106 actions are legal. Before the loss function computes probabilities via softmax, the logits of illegal actions are set to -10^9, which effectively forces their probability to zero. This ensures the model only assigns probability mass to actions that are actually available, and the loss function does not penalize the model for its predictions on illegal actions.
Total parameters: 1.77M (most of which are in the first layer: $2840 \times 512 = 1.45$M weights).

4.3 Experiment 0: Initial Training (800 games)

The V1 model trained on 800 games achieved approximately 55.9% top-1 validation accuracy. Training loss reached 0.42 while validation loss plateaued at 1.57, a large gap indicating severe overfitting. This is expected with a relatively small dataset (800 games, 84,331 decisions) and a 1.77M-parameter model.

For context, the random-legal reference strategy achieves approximately 29% and the most-frequent-legal strategy is 44% (Section 3.3), so the model outperforms trivial strategies even at this early stage.

4.4 Experiment 1: Regularization (800 games)

Each regularization technique was added incrementally on top of the previous best configuration:

- **Early stopping** halts training when validation loss stops improving (patience of 7 epochs), preventing the model from continuing to overfit.
- **Dropout** [17] randomly disables 50% of neurons during each training step, forcing the network to learn redundant representations rather than relying on individual neurons.
- **Weight decay** ($\lambda = 10^{-4}$) adds an L2 penalty on weight magnitudes to the optimizer, discouraging overly large weights and encouraging simpler solutions.
- **Batch normalization** [5] normalizes layer activations to zero mean and unit variance within each batch, stabilizing training and acting as a mild regularizer.
- **Learning rate scheduling** (ReduceLROnPlateau) halves the learning rate when validation loss plateaus for 5 epochs, allowing finer weight adjustments near convergence.

Config	Val Acc	Train Loss	Val Loss	Best Epoch
0: No regularization	55.9%	0.42	1.57	—
1a: Early stopping	55.57%	0.869	0.935	5
1b: + Dropout(0.5)	55.92%	0.902	0.920	22
1c: + Weight decay	55.84%	0.902	0.918	22
1d: + BatchNorm	56.10%	0.836	0.913	23
1e: + LR scheduling	56.11%	0.824	0.913	23
1f: All combined	**56.25%**	**0.855**	**0.913**	**18**

Table 4.2: Regularization ablation results.

The most significant effect of regularization is visible in the overfitting gap (validation loss minus training loss). Without regularization (Experiment 0), the model memorizes the training set (train loss 0.42) while validation loss climbs to 1.57, producing a gap of 1.15. Early stopping is the most impactful technique: by halting training at epoch 5, it reduces this gap to 0.066 (at best epoch). Dropout further narrows the gap to 0.018 (at best epoch),mainly because training became harder (train loss rose from 0.869 to 0.902), though validation loss also improved slightly (0.935 to 0.920). The Best Epoch column shows that dropout also required substantially more training (22 epochs vs. 5), since the model can no longer memorize quickly with half its neurons randomly disabled. Despite this large reduction in overfitting, validation accuracy improved by less than 1 percentage point across all configurations (55.57% to 56.25%). The overall picture is that regularization controlled overfitting but did not translate into meaningful accuracy gains.

Note that the differences between configurations are small and may be partially due to noise (single-run experiments). The main takeaway is the pattern (regularization helps training dynamics but not accuracy) rather than the specific ranking.

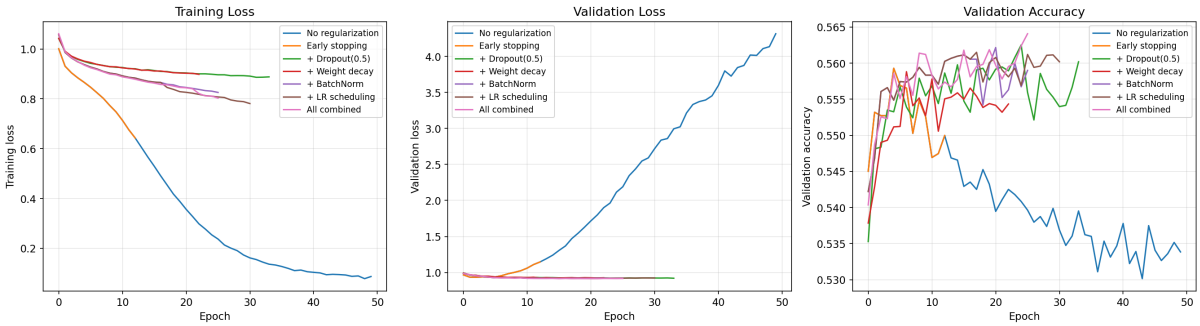

Figure 4.1: Learning curves (train/val loss) for selected regularization configurations.

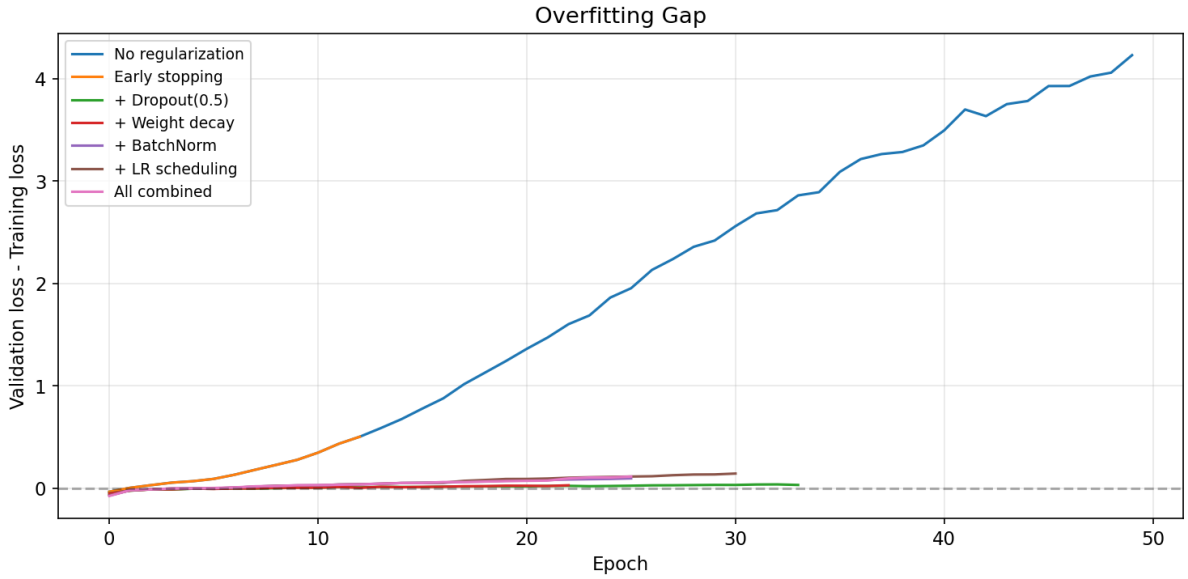

Figure 4.2: Overfit gap comparison across configurations.

4.5 Experiment 2: Data Scaling (up to 6,400 games)

With the regularization configuration from Experiment 1 (Section 4.4), training data was scaled from 800 to 6,400 games:

Games	Decisions	Val Acc	Top-3	Train Loss	Val Loss	Best Epoch
800	85,207	56.86%	89.72%	0.864	0.893	13
1,600	167,820	57.37%	90.08%	0.813	0.870	28
4,000	420,307	58.83%	91.21%	0.791	0.830	64
6,400	**538,031**	**58.87%**	**91.22%**	**0.803**	**0.822**	**50**

Table 4.3: Data scaling results.

Increasing the dataset from 800 to 6,400 games improved validation accuracy by approximately 2 percentage points. Most of this gain occurred between 800 and 4,000 games, with virtually no improvement from 4,000 to 6,400 games (58.83% vs 58.87%). This plateau probably suggests the model has reached a performance limit with this architecture and encoding.

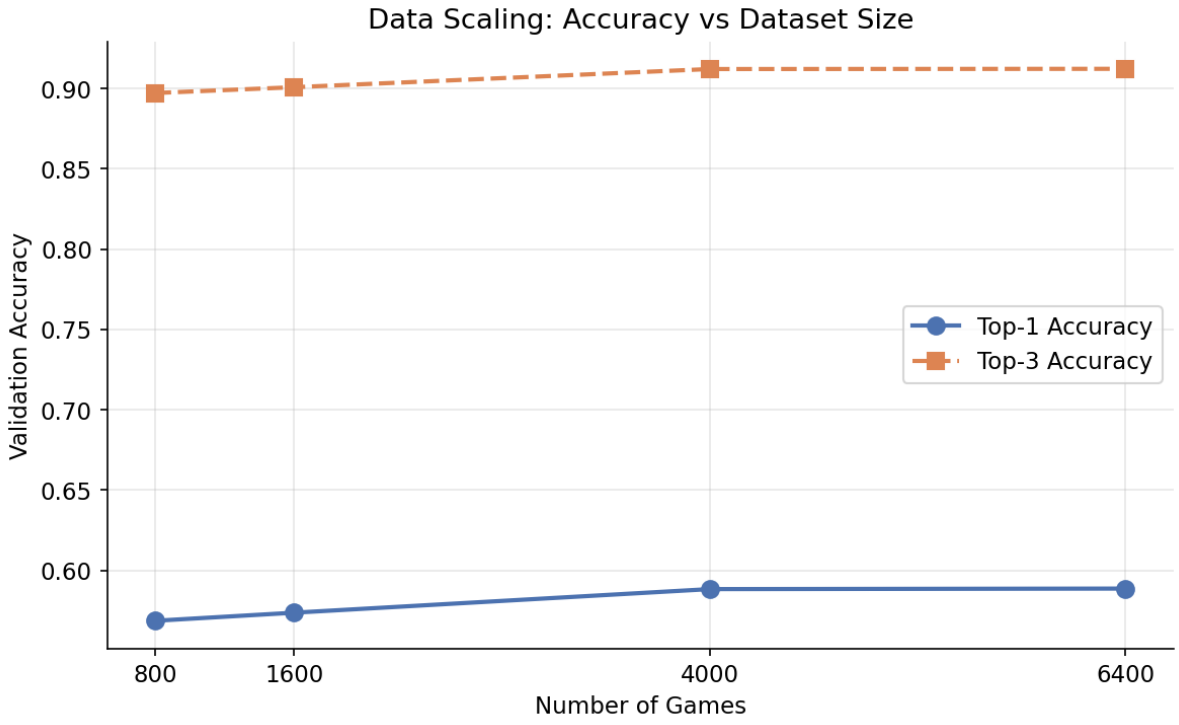

Figure 4.3: Validation accuracy vs. dataset size - the main data scaling result.

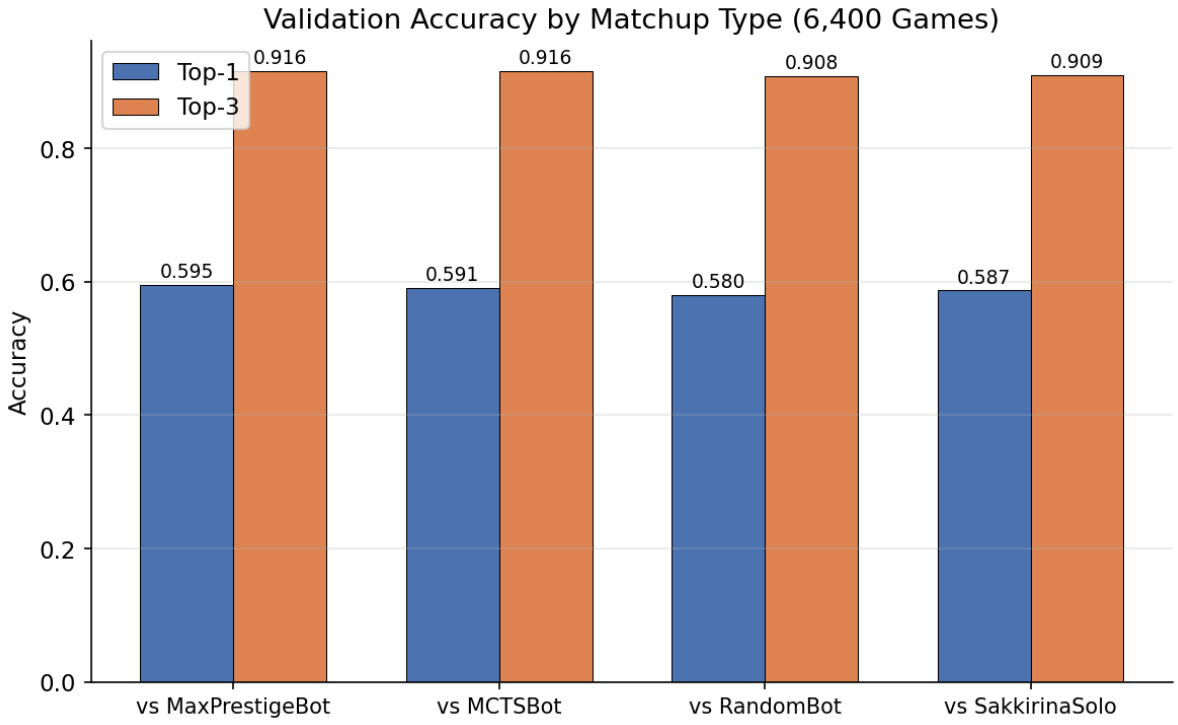

Figure 4.4: Accuracy by matchup type on the full dataset.

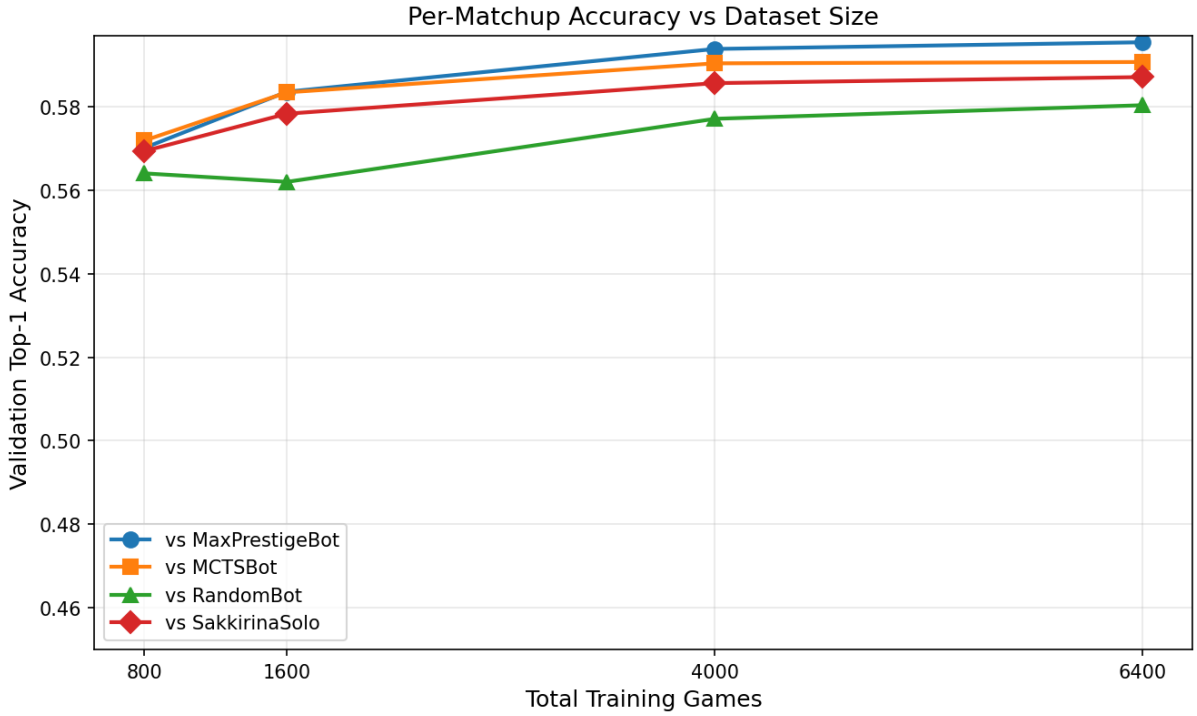

Figure 4.5: Per-matchup accuracy across dataset sizes.

4.6 Teacher Information Gap

The data scaling results show diminishing returns beyond approximately 4,000 games, suggesting the accuracy ceiling is not caused by insufficient data. A likely explanation lies in the difference between what the teacher and the student can observe. The teacher (SakkirinaSolo) is an MCTS bot that simulates future game states during search, effectively evaluating the long-term consequences of each move. The student, by contrast, learns a direct mapping from the current visible game state to an action. Some teacher decisions therefore depend on information not available to a model that only observes the current game state without performing search. This information gap likely places a ceiling on imitation accuracy regardless of model capacity or data volume, and motivates the architectural and methodological changes explored in the following chapters.

5 | Improved Representations

The initial system (Chapter 4) plateaued at ∼59% accuracy with diminishing returns from regularization and data scaling. Since adding more data and controlling overfitting both failed to improve accuracy, the next question is whether the way the game state is represented limits what the model can learn, perhaps the flat 2,840-dimensional feature vector loses information or relationships that a different representation could preserve. This chapter explores that question, first through feature engineering on the V1 architecture, then through a redesigned V2 architecture with structured encoding and pointer-based action selection.

5.1 Experiment 3: Feature Engineering

Three feature engineering approaches were evaluated on the V1 architecture:

Config	Val Acc	Top-3	Parameters	Change
V1 (Exp 2)	58.87%	91.22%	1,773,674	—
3a: + Normalization	58.86%	91.21%	1,773,674	−0.01%
3b: + Compact slots	58.78%	91.09%	1,077,354	−0.09%
3c: + Sorted cards	51.56%	86.69%	1,077,354	−7.31%

Table 5.1: Feature engineering results.

Normalization (3a) standardized scalar features (such as coins, prestige, pile counts) to zero mean and unit variance across the training set. This had no measurable effect. The V1 features are mostly one-hot encodings already in a reasonable range; the 15 global scalar features that benefit from normalization make up only 0.5% of the 2,840-dimensional input.

Compact choice slots (3b) reduced the number of choice slots in the input encoding from 64 to 24. The original 64-slot allocation was chosen conservatively, but the maximum number of pending choice options observed in the dataset is 21, meaning at least 43 slots were always zero-padded. Reducing to 24 slots shrank the input from 2,840 to 1,480 dimensions and cut parameters by 39% with negligible accuracy loss (−0.09 percentage points). This became the default for subsequent experiments.

Card sorting (3c) caused a 7.3% drop. This failure highlighted the practical consequence of the positional action space: actions reference card **positions**, not card **identities**. Action index 0 means "play the card in hand slot 0," so sorting cards changes which features appear in which slots without updating the action labels, destroying the learned correspondence. The overall 7.3% drop also understates the effect on position-dependent action types, since action types that do not reference card positions (such as END_TURN and CALL_PATRON) are unaffected by sorting, diluting the measured drop. This finding directly motivated the V2 architecture.

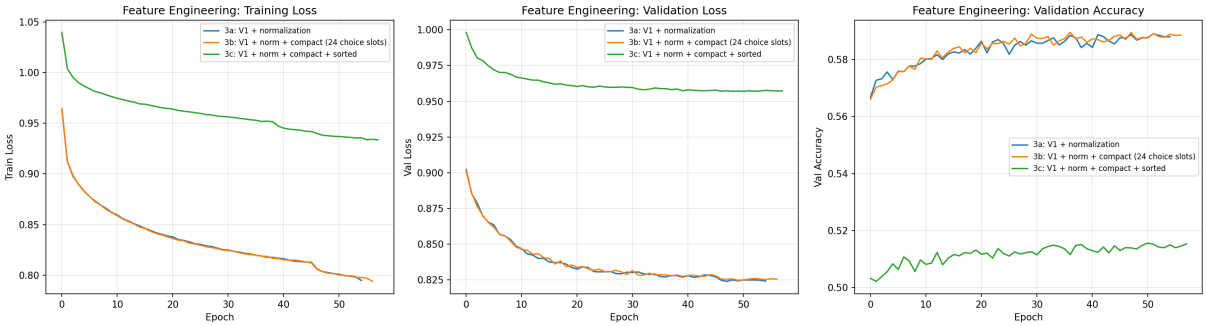

Figure 5.1: Learning curves for all three feature engineering variants.

5.2 Experiment 4: V2 Architecture - Pointer-Based Action Head

5.2.1 Motivation

Experiment 3c (Section 5.1) demonstrated that input processing that ignores card order (desirable for card sets, since hand order has no gameplay significance) is incompatible with the positional action encoding used in V1. To address this, the V2 architecture uses a **pointer-based action head** inspired by Pointer Networks [18], where each action is scored by content rather than position.

5.2.2 Architecture

Unlike V1, which concatenates all card features into a single flat vector, V2 keeps cards separated by zone and processes them individually. The architecture has four stages: embedding, pooling, trunk processing, and pointer-based action scoring.

Stage 1: Embedding. Each item's raw features are transformed into a richer learned representation through a linear layer followed by ReLU. Hand and tavern cards use the same 17-dimensional vector from V1 (8-dim deck one-hot + 6-dim type one-hot + cost + HP + taunt). Agent cards use a 19-dimensional vector (the 17 card features plus HP and taunt as separate agent-state fields). Choice options use a 34-dimensional vector (3 scalar fields + 17 card features + 14-dim effect one-hot). Global features are a 24-dimensional vector (player resources, pile counts, patron states) projected to the embedding dimension through a separate linear layer. Concretely, for a hand card with raw features $x \in \mathbb{R}^{17}$, the embedding is $e = \text{ReLU}(W_{\text{hand}} \cdot x + b_{\text{hand}})$, where $W_{\text{hand}} \in \mathbb{R}^{64\times17}$ is a learned weight matrix and $b_{\text{hand}} \in \mathbb{R}^{64}$ is a learned bias. The purpose of this transformation is to convert the raw features (a mix of unrelated one-hot segments and scalars that are not directly comparable) into a common vector space where cards with similar roles end up with similar vectors. This makes the downstream dot-product scoring meaningful: two cards that serve a similar purpose will have similar embeddings and thus receive similar scores. Each zone (hand, tavern, agents, choices) has its own embedding layer, allowing the network to learn zone-specific representations. For example, a Gold card in the hand (about to be played) may need a different representation than the same card in the tavern (available for purchase).

Stage 2: Pooling. The trunk needs a fixed-size summary of each zone, but the number of cards varies (a hand might have 3 or 12 cards). Pooling compresses a variable-length set of card embeddings into a single fixed-size vector. Two variants were tested: **mean pooling**, which averages all card embeddings in a zone (giving equal weight to every card), and **attention pooling** [9], which computes a learned weighted average. In attention pooling, a learnable query vector $q_{\text{pool}} \in \mathbb{R}^{64}$ is compared against each card embedding via dot product to produce attention weights: cards that are more "relevant" (as determined by training) receive higher weight in the summary. Padding slots (representing empty card positions) are masked out so they do not influence the summary.

Stage 3: Trunk. The five zone summaries (hand, tavern, own agents, enemy agents, choices—each 64-dimensional) and the global features (projected to 64 dimensions) are concatenated into a single 384-dimensional vector. This is projected to a 256-dimensional trunk representation and processed through

two residual blocks [4]. Each residual block applies two linear layers with batch normalization and ReLU activations, then adds the block's input back to its output (output $= x + f(x)$). The trunk output is a 256-dimensional vector that represents the model's overall understanding of the current game state.

Stage 4: Pointer-based action scoring. This is the key difference from V1. Instead of a single linear layer mapping the trunk to 106 action logits, each card-based action is scored individually by comparing a learned query against the card's embedding. For each zone (hand, tavern, etc.), a separate linear layer projects the 256-dimensional trunk into a 64-dimensional query vector q. The score for action i is the scaled dot product between this query and the card's embedding:

$$\text{score}_i = \frac{q^\top e_i}{\sqrt{d}}$$

where $q \in \mathbb{R}^{64}$ is the query derived from the trunk (representing "what kind of card should be acted on given the current game state"), $e_i \in \mathbb{R}^{64}$ is the embedding of the i-th card in that zone (representing "what kind of card this is"), and $\sqrt{d} = \sqrt{64} = 8$ is a scaling factor that prevents the dot products from becoming too large. A high score means the model considers this card a good candidate for the action; a low score means it does not. Since the score depends on the card's content (its embedding) rather than its position in the array, identical cards in different positions receive identical scores.
Patron actions and END_TURN are not card-based—patrons are always presented in a fixed alphabetical order, and END_TURN is a single global action. These are scored by conventional linear heads applied directly to the trunk:

$$\text{patron\_scores} = W_{\text{patron}} \cdot \text{trunk} + b_{\text{patron}} \quad \text{(5 scores)}$$

$$\text{end\_turn\_score} = W_{\text{end}} \cdot \text{trunk} + b_{\text{end}} \quad \text{(1 score)}$$

All scores are concatenated into the 106-dimensional logit vector in action space order (hand: 17 + tavern: 5 + patrons: 5 + activate: 7 + attack: 7 + choice: 64 + end_turn: 1 = 106) and masked as in V1. The legal action mask resolves any residual ambiguity: if two identical cards are both legal, both receive the same score, and selecting either is equally correct.

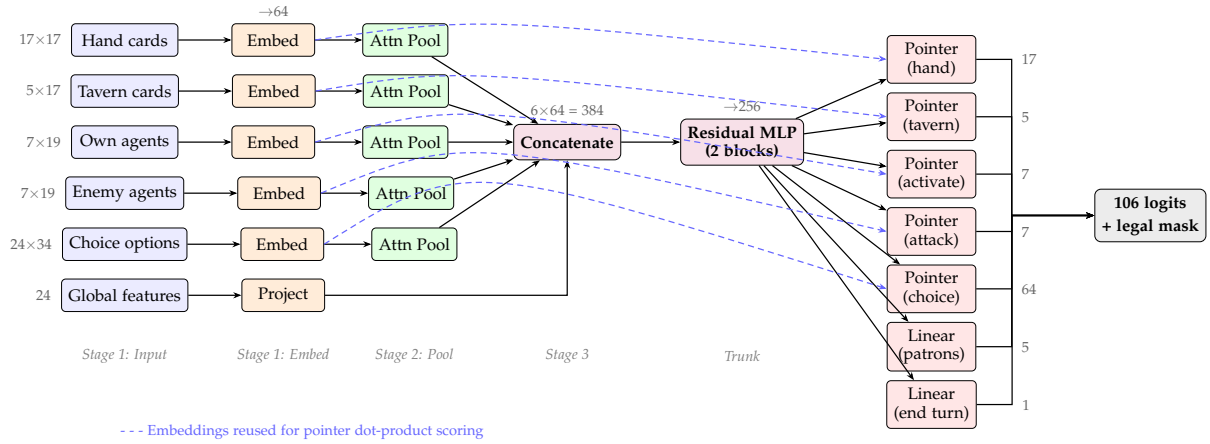

Figure 5.2: PolicyNetworkV2 architecture. Solid arrows show the forward pass; dashed arrows indicate that zone embeddings from Stage 1 are reused for pointer scoring in the action heads. Each pointer head computes a scaled dot product between a query derived from the trunk and each card embedding: $\text{score}_i = q^\top e_i / \sqrt{d}$, where $q \in \mathbb{R}^{64}$ is the query, $e_i \in \mathbb{R}^{64}$ is the card embedding, and $d = 64$ is the embedding dimension. Patron and end-turn actions use conventional linear heads since they are not card-based.

5.2.3 Results

Four V2 configurations were compared against the V1 baseline, varying the pooling method (mean vs. attention) and model size (small, default, large).

Config	Val Acc	Top-3	Train Loss	Val Loss	Parameters	Best Epoch
V1 (Exp 2)	58.87%	91.22%	0.803	0.822	1,773,674	50
4a: Default V2, mean pool	59.24%	91.02%	0.807	0.823	456,774	17
4b: Default V2, attention pool	59.24%	90.92%	0.805	0.820	457,094	18
4c: Small V2	59.47%	91.12%	0.802	0.821	84,422	33
4d: Large V2	59.37%	90.97%	0.805	0.821	2,326,278	12

Table 5.2: V2 architecture results.

z = zone embedding dimension, t = trunk dimension, b = number of residual blocks. Default (4a, 4b): z=64, t=256, b=2. Small (4c): z=32, t=128, b=1. Large (4d): z=128, t=512, b=3.

Three findings emerge:

1. **Mean vs. attention pooling** produced virtually identical results (59.24% for both).
2. **Model size had no effect.** The small model (84K parameters) matched the large model (2.3M parameters): 59.47% vs. 59.37%. This 27x parameter difference produced no meaningful accuracy change, suggesting model capacity is not the limiting factor.

3. **V2 vs. V1:** The pointer-based architecture matches V1 accuracy (59.24% vs. 58.87%, within single-run noise). The primary achievement is that V2 successfully decouples action scoring from card positions while maintaining accuracy. As a side effect, pointer heads reuse card embeddings rather than learning a separate output weight per action, reducing parameters from 1.77M to 84K (21x).

Note: 4b (default V2, attention pool) is the configuration used in all subsequent experiments.

6 | Beyond Behavioral Cloning

The experiments in Chapters 4–5 established that neither model capacity nor representation changes break the ∼59% accuracy plateau. The next step is to test the model in live play, where the gap between offline accuracy and actual win rate reveals how much the accuracy plateau matters in practice, and to try DAgger, which addresses the possibility that the training data does not cover the states the model encounters during actual games.

6.1 Live Bot Deployment

The trained model was deployed as a live bot (LT_NN) integrated with the C# game engine.

Against SakkirinaSolo (50 games), LT_NN achieved a 12% win rate (95% CI: 4.5%–24.3%). The gap between ∼59% offline accuracy and 12% live win rate illustrates the compounding-error problem of behavioral cloning [14]. The model is correct on 59% of individual decisions, but errors accumulate: with approximately 9 decisions per turn, the probability of making no mistakes in an entire turn is $0.59^9 \approx 0.9\%$. Each mistake can push the game into a state the model has not seen during training, making subsequent predictions less reliable. Not every mistake is strategically important, but over the course of a full game, enough errors compound to reduce the win rate far below what the per-decision accuracy would suggest.

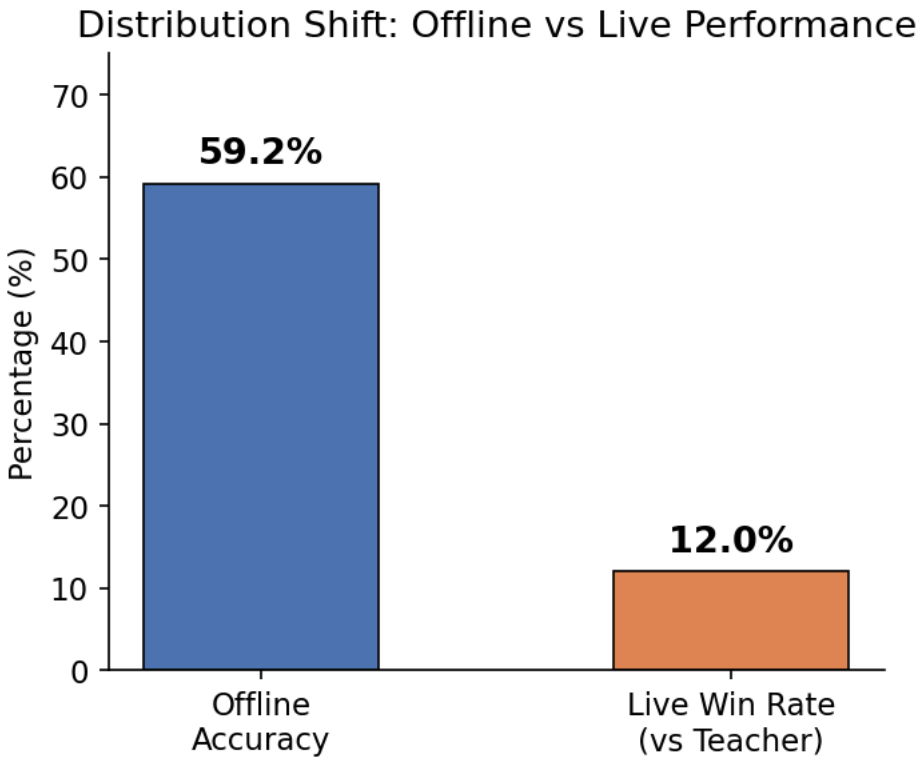

Figure 6.1: Offline accuracy vs. live win rate, illustrating the distribution shift gap.

6.2 Experiment 5: DAgger

The ∼59% accuracy / 12% win rate gap motivates DAgger [14], which addresses distribution shift by collecting teacher corrections on states visited by the learned policy. During data collection, the learner makes the actual game decisions, but at each decision point the teacher independently evaluates the same state and records what it would have chosen. This produces training data from states the learner actually visits rather than states the teacher visits.

Each DAgger iteration follows the same cycle: (1) play games with the current learner to collect teacher-labeled data from learner-visited states, (2) aggregate the new data with the existing dataset, (3) retrain the model from scratch on the combined dataset, so the policy is not biased toward earlier data.

Iteration	Learner Model	New Samples	Cumulative DAgger	DAgger Fraction
1	BC-only model	57,029	57,029	9.6%
2	Iter 1 model	55,616	112,645	17.3%
3	Iter 2 model	56,411	169,056	23.9%

Table 6.1: DAgger data collection (3 iterations).

Each iteration: 500 games, LT_NN as Player 1 vs. SakkirinaSolo as Player 2, 10s turn timeout. Note: iteration 1 used a 30s timeout, allowing the teacher deeper MCTS search - this discrepancy is a limitation.

Config	Val Acc	Top-3	Best Epoch	Total Samples
BC-only	59.24%	90.92%	18	537,582
+ DAgger iter 1	59.25%	91.09%	13	594,611
+ DAgger iter 2	59.58%	91.10%	17	650,227
+ DAgger iter 3	59.47%	90.99%	17	706,638

Table 6.2: DAgger offline accuracy.

Three DAgger iterations cumulatively adding 169,056 samples (23.9% of the combined dataset) produced no meaningful change in offline accuracy. Validation accuracy fluctuated within a 0.34 percentage point range, consistent with training noise.

All four models were evaluated using the tournament runner with $n = 100$ games per opponent (50 per side), 4 threads, 10s timeout:

Opponent	Pre-DAgger	Iter 1	Iter 2	Iter 3
RandomBot	100%	100%	100%	100%
MaxPrestigeBot	62%	58%	69%	56%
BeamSearchBot	17%	21%	28%	16%
DecisionTreeBot	30%	36%	38%	33%
SakkirinaSolo	11%	9%	12%	3%
Mean (excl. Random)	**30.0%**	**31.0%**	**36.8%**	**27.0%**

Table 6.3: DAgger win rate evaluation ($n = 100$ per opponent).

With $n = 100$ games per opponent, win rate estimates have a standard error of approximately 3-5 percentage points, meaning most cross-iteration differences in the table fall within noise. Iter 2 showed

the best results across all opponents, but iter 3 regressed below the pre-DAgger model (mean 27.0% vs 30.0%), demonstrating no consistent improvement trend.

The persistent ∼59% offline accuracy plateau and lack of consistent win rate improvement across DAgger iterations raise a question about the applicability of DAgger to this setting. A straightforward explanation connects to the teacher information gap identified in Section 4.1: if the accuracy plateau is caused by the student lacking access to the teacher's search results, then DAgger cannot help - it provides examples from different game states, but the student still cannot observe what the teacher uses to make decisions. DAgger addresses which states appear in the training data (distribution shift), but the bottleneck may be what information is available at each state (information gap). Under this explanation, no amount of additional DAgger data would improve accuracy, because the missing information is not in the training labels but in the teacher's decision process.

From a theoretical perspective, DAgger's formal convergence guarantees [14] assume learners that can find globally optimal policies on the aggregated dataset—a condition that neural networks, with their non-convex loss landscapes, do not satisfy. As [8] observe, existing theoretical results "do not capture the empirical success of FTL (e.g., DAgger) used in conjunction with complex policy parameterizations like neural networks for which the loss functions are non-convex." However, DAgger has been successfully applied with neural networks in other domains (e.g., robotics, autonomous driving), so this theoretical gap is a caveat rather than a complete explanation for the negative result observed here.

Additionally, the teacher quality ceiling limits the potential benefit of corrections: the teacher wins approximately 84% of training games, and teacher corrections on learner-visited states may not be significantly better than the learner's own decisions in many game situations. The DAgger data constitutes up to 24% of the combined dataset, yet produces no measurable improvement. However, only 1,500 DAgger games were collected (3 iterations of 500), with a single teacher and no hyperparameter tuning for the mixed-distribution training, so it is not possible to determine whether the limitation is the approach itself or insufficient data and iterations. The negative result here may be specific to this particular learner, teacher, and game.

7 | Evaluation and Discussion

This chapter presents the final evaluation of the trained model: test-set accuracy with baseline comparisons, per-action-type breakdown, ranking quality analysis, component ablation, inference speed, a tournament against competition bots, and patron selection.

7.1 Final Test-Set Evaluation

On the held-out test set (640 games, 67,617 decisions), the final checkpoint (DAgger iter 2, the same checkpoint used in the tournament evaluation) achieves **60.41% top-1 accuracy** and **91.48% top-3 accuracy**. The test-set accuracy (60.41%) is slightly higher than the validation accuracy ($\sim$59%) observed during training, which is expected given normal sampling variation and confirms no overfitting to the validation set. The top-3 accuracy indicates that the correct action is almost always among the model's top 3 predictions.

In subsequent discussion, the $\sim$59% figure refers to the validation accuracy plateau observed consistently across Experiments 4–5 (the pattern that establishes the ceiling), while 60.41% refers specifically to this test-set evaluation.

7.2 Per-Action-Type Accuracy

The target labels throughout this analysis are the teacher's actions - the model's predictions are compared against the teacher's exact choices at each decision point.

Baseline comparison. To contextualize the 60.41% overall accuracy, Table 7.1 compares the model against the trivial baselines introduced in Section 3.3:

Strategy	Accuracy
Random legal	28.86%
Most frequent legal	44.01%
NN model	**60.41%**

Table 7.1: Baseline comparison (test set).

The model achieves 16.4 percentage points above the strongest trivial baseline. The most-frequent-legal baseline always picks the globally most common legal action, which is nearly always PLAY_CARD slot 0 (the most frequent action in the dataset at 27.6%). This means it never selects BUY_CARD, CALL_PATRON, ACTIVATE_AGENT, or ATTACK actions, scoring 0% on those types. The model, by contrast, achieves non-trivial accuracy across all action types (Table 7.2).

Action Type	Count	% of Total Actions	Top-1	Top-3
END_TURN	7,589	11.2%	92.62%	99.38%
ATTACK	340	0.5%	73.53%	96.47%
ACTIVATE_AGENT	626	0.9%	70.61%	95.85%
CHOICE	7,479	11.1%	67.83%	89.89%
BUY_CARD	5,647	8.4%	61.20%	88.10%
PLAY_CARD	39,606	58.6%	56.37%	92.10%
CALL_PATRON	6,330	9.4%	35.94%	82.27%

Table 7.2: Accuracy by action type (test set).

Accuracy varies across action types. Actions with unambiguous triggers (END_TURN: 92.6%) are nearly perfectly predicted. Strategically complex actions requiring timing judgment (CALL_PATRON: 35.9%) are the model's primary weakness, the confusion matrix (Figure 7.2) shows CALL_PATRON decisions frequently predicted as PLAY_CARD (1,105 cases, 17.5% of all CALL_PATRON decisions) or BUY_CARD (916 cases, 14.5%), suggesting the model identifies that "do something" is appropriate but struggles with the patron-vs-card priority.

PLAY_CARD is the most frequent action type (58.6% of all decisions). Its 56.4% accuracy heavily influences the overall metric: the model must select which card to play from potentially 5+ hand cards. This is a within-type selection problem.

Top-3 accuracy is uniformly high (82-99%), indicating the model learns a reasonable action ranking even when it does not select the teacher's exact first choice.

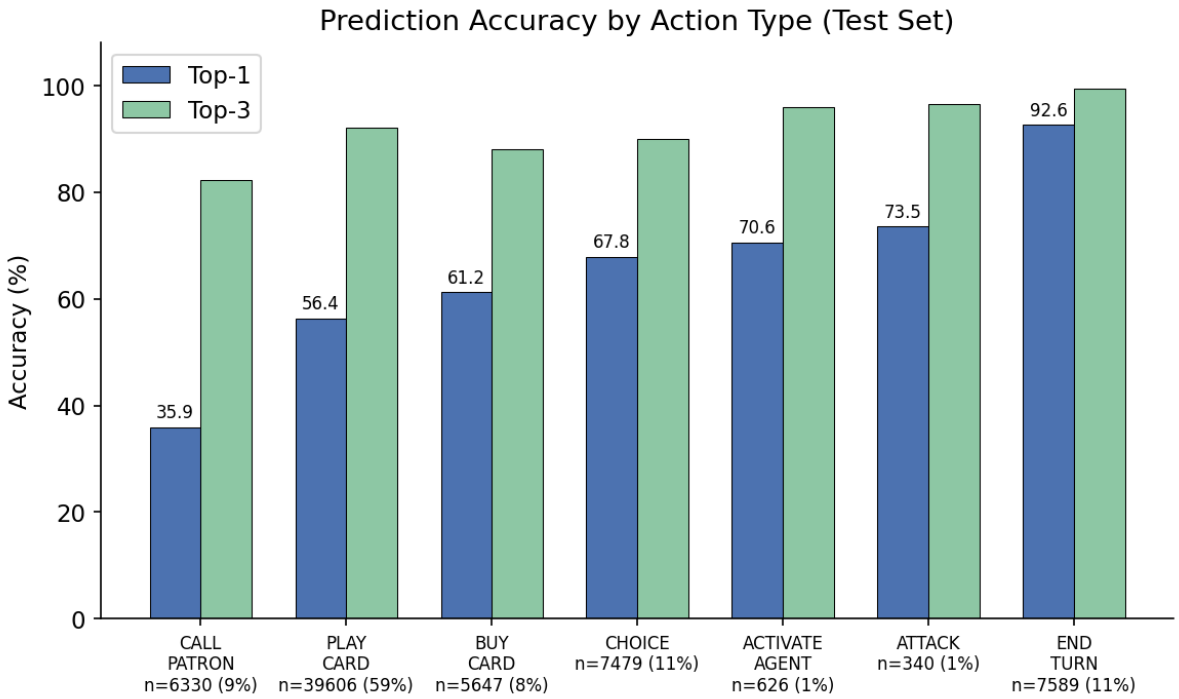

Figure 7.1: Top-1 and Top-3 accuracy by action type, ordered by difficulty (hardest on the left).

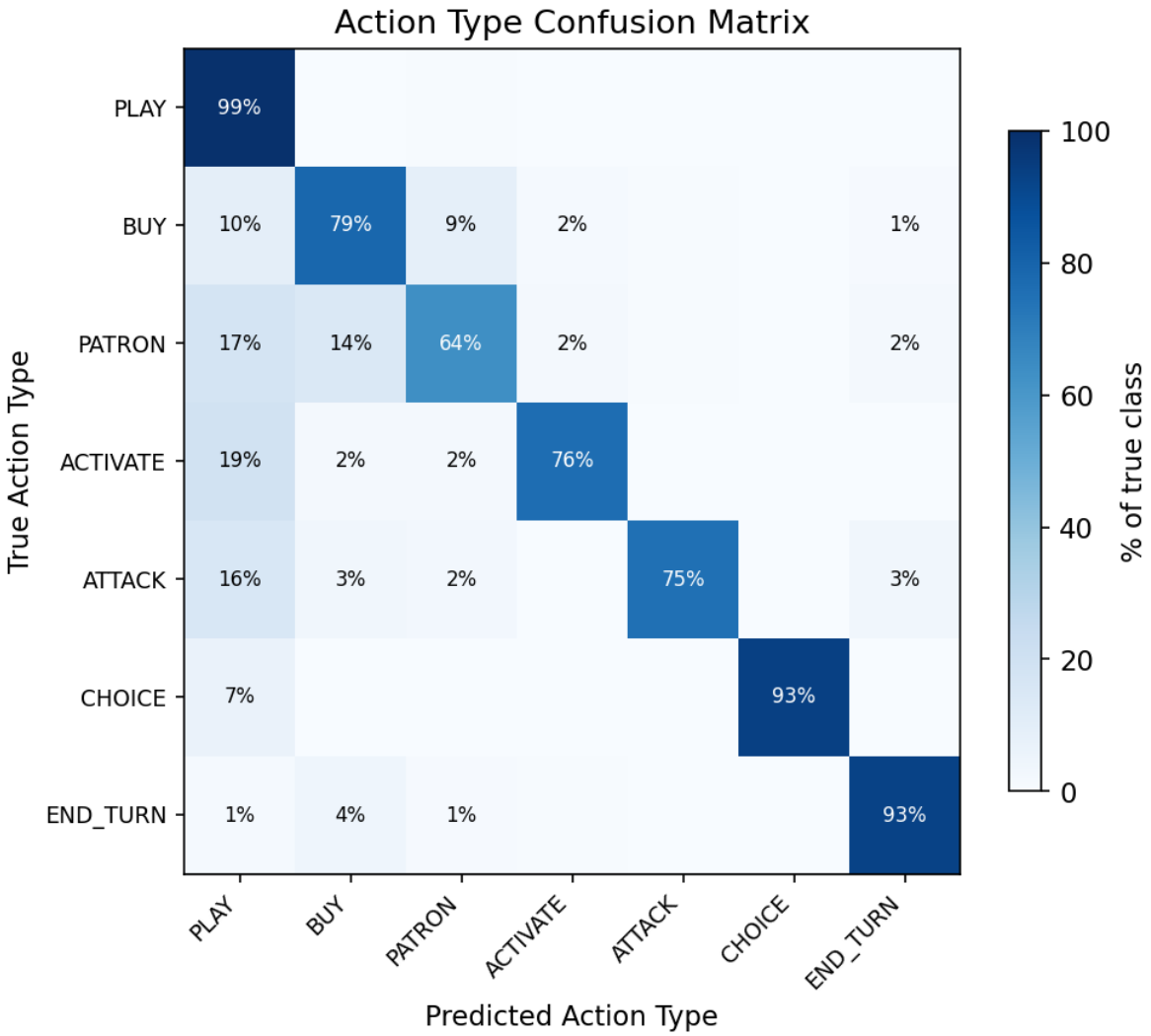

Figure 7.2: Action type confusion matrix. Each row is a true action type (what the teacher chose) and each column is the model's prediction. Values are percentages of the row total, so each row sums to 100%. The matrix groups all 106 actions into 7 types, showing cross-type errors (e.g., predicting PLAY_CARD when the teacher chose CALL_PATRON) but not within-type errors (e.g., playing the wrong card from the hand).

7.3 Ranking Quality (ROC-AUC)

Using the legal-restricted ROC-AUC metric defined in Section 3.3, the model achieves a macro AUC of 0.892 and a weighted AUC of 0.854.

Action Type	AUC
END_TURN	0.999
PLAY_CARD	0.980
CALL_PATRON	0.973
BUY_CARD	0.961
ATTACK	0.952
ACTIVATE_AGENT	0.946

Table 7.3: AUC by action type (legal-restricted).

At the action-type level, all types achieve AUC above 0.94, meaning the model ranks action types well even when individual selections within a type are harder. The more challenging question is *which specific action within a type* (for example, which of 5 hand cards to play). At the individual action level, PLAY_CARD slots 1–4 have AUC 0.66–0.78, reflecting the difficulty of selecting among multiple hand cards. END_TURN is near-perfect (0.999).

A weighted AUC of 0.854 is consistent with the $\sim$59% top-1 accuracy and approximately 91% top-3 accuracy observed throughout the experiments: the model produces reasonable rankings but faces ambiguity among the top candidates. This is an important finding for interpreting the $\sim$59% accuracy: when the model selects incorrectly, it tends to pick a near-miss rather than a random action. The model has learned meaningful card evaluation (it understands which actions are reasonable) but struggles to distinguish between the top 2-3 candidates, which is where the teacher's search advantage likely matters most.

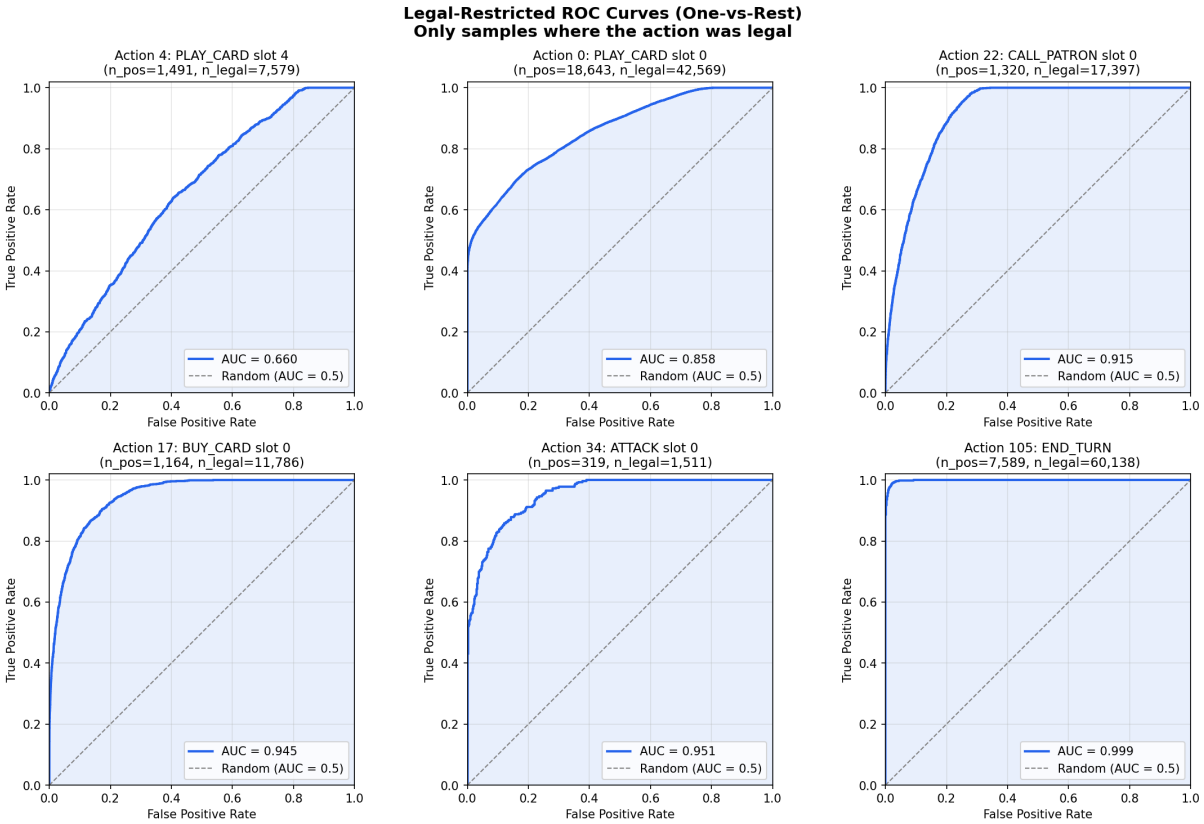

Figure 7.3: ROC curves for six representative actions, from near-random (PLAY_CARD slot 4, AUC 0.66) to near-perfect (END_TURN, AUC 0.999).

7.4 Component Ablation

Each input zone was zeroed out (all features and legal action flags for that zone set to zero) to quantify the model's reliance on each type of information. The accuracy drop is the difference between the baseline and ablated accuracy (baseline: 60.41%); the relative drop expresses this as a percentage of the baseline accuracy.

Zone Removed	Accuracy	Accuracy Drop	Relative Drop
None (baseline)	60.41%	—	—
Hand cards	28.37%	−32.05 pp	−53.0%
Choice options	52.91%	−7.50 pp	−12.4%
Global features	53.53%	−6.88 pp	−11.4%
Tavern cards	54.85%	−5.56 pp	−9.2%
Own agents	59.59%	−0.82 pp	−1.4%
Enemy agents	60.01%	−0.41 pp	−0.7%

Table 7.4: Component ablation (test set).

Hand cards are overwhelmingly the most important input (53.0% relative drop), consistent with PLAY_CARD being 58.6% of decisions. Choice context, global features (coins, power, prestige), and tavern cards each contribute 9–12%. Agent board state contributes less than 1.3 percentage points combined (0.82% + 0.41%), since agent-related actions (ACTIVATE_AGENT, ATTACK) make up only 1.4% of decisions, limiting the model's incentive to rely on this information. This pattern suggests the model has primarily learned card selection (which card to play, which to buy) but has not learned to use board state for strategic planning, consistent with the overall picture that the model captures the routine aspects of play but struggles with the strategic decisions where the teacher's search advantage matters most.

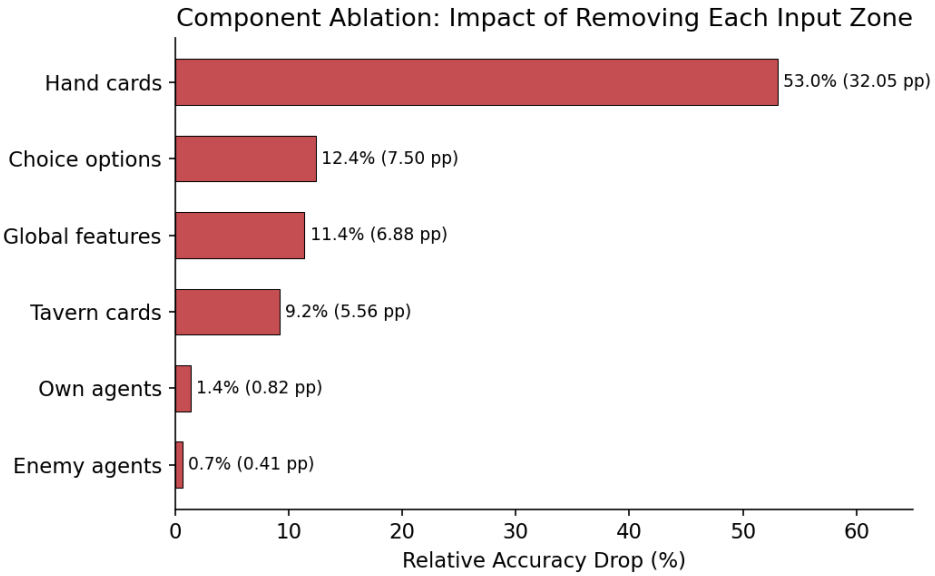

Figure 7.4: Component ablation: relative accuracy drop when each input zone is removed.

7.5 Decision-Time Measurements

Inference latency was measured over 1,000 single-sample forward passes on the deployed model.

Component	Time
State encoding (JSON to tensors)	0.058 ms
Neural network forward pass (mean)	5.411 ms
Neural network forward pass (median)	4.794 ms
P95 latency	7.90 ms
P99 latency	9.45 ms
Total per decision	**5.47 ms**
Throughput	**183 decisions/sec**

Table 7.5: Inference latency (RTX 2060, 457K parameters, CUDA).

P95 and P99 latencies represent the 95th and 99th percentiles—95% and 99% of decisions complete within these times respectively. At approximately 5 ms per decision and $\sim$105 decisions per game, LT_NN completes all decisions in approximately 0.5 seconds total.

7.6 Tournament Evaluation

The DAgger iter 2 model (best-performing in the DAgger evaluation) was evaluated against 11 bots from the COG 2025 competition, with 500 games per matchup (5 rounds of 100 games, 50 per side, 4 threads, 10s timeout).

The tournament included bots from three categories: **heuristic-based** bots that follow hand-crafted rules, **search-based (MCTS)** bots that simulate future game states to evaluate moves, and **neural-network-based** bots using learned policies.

Opponent	LT_NN Win Rate	Bot Type
Vei	99.8%	Heuristic
ZMyBot	67.6%	Heuristic
CouncilOfTwo	60.4%	Heuristic
AIFBotMCTS	35.8%	MCTS
EBot_C	32.8%	Heuristic
NAgent	31.4%	Neural Network
EBot_F	24.2%	Heuristic
BestMCTS3	15.2%	MCTS
SakkirinaSolo	14.2%	MCTS
HQL_Plus_BOT	12.6%	MCTS
Sakkirina	12.0%	MCTS
Overall	**36.9% (2,030 / 5,500)**	

Table 7.6: Tournament results ($n = 500$ per opponent).

Performance can be grouped into three tiers:

- **Winning**: Vei, ZMyBot, CouncilOfTwo
- **Intermediate**: AIFBotMCTS, EBot_C, NAgent, EBot_F
- **Losing**: BestMCTS3, SakkirinaSolo, HQL_Plus_BOT, Sakkirina

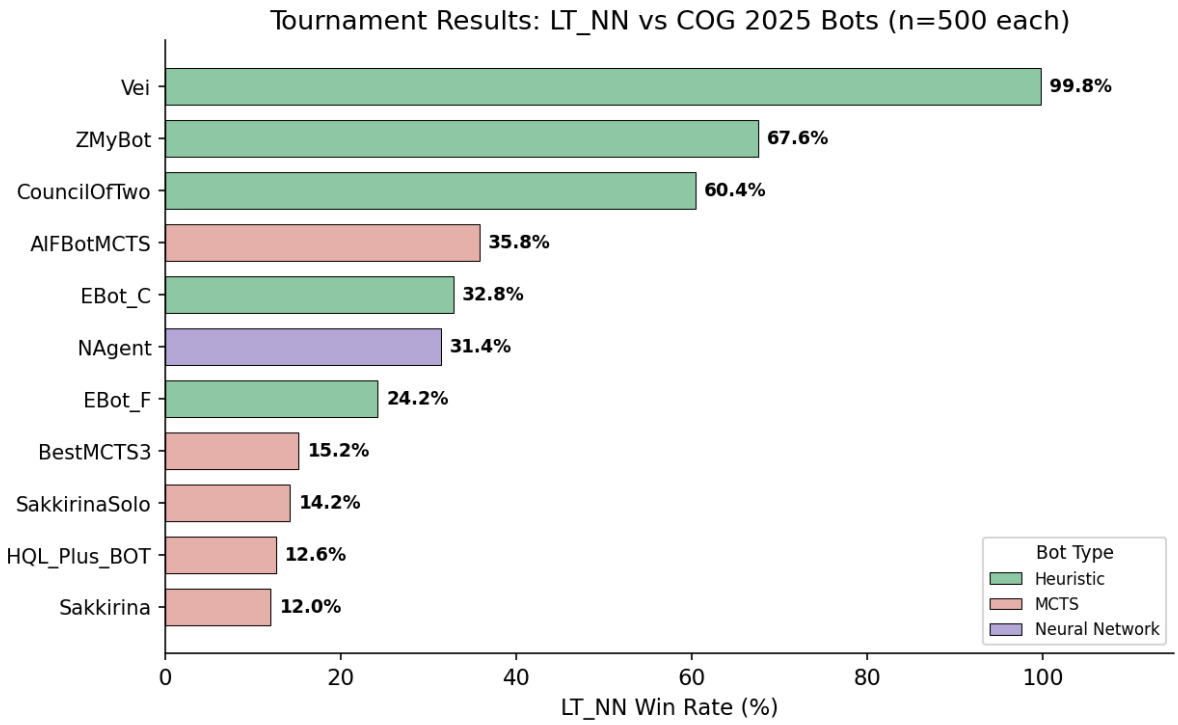

Figure 7.5: Tournament win rates by opponent, colored by bot type (Heuristic, MCTS, Neural Network).

7.7 Patron Selection

Each game of Tales of Tribute begins with a patron draft, where both players alternate selecting patrons until four are active. Different patron combinations create different strategic landscapes, so this section examines whether the teacher performs better with certain patrons than others.

The teacher's win rate was measured per active patron across all 12,800 games in the full collected dataset (before the 6,400-game subset was selected for training). These win rates reflect the teacher's performance in games where each patron was active, regardless of which player selected it during the draft.

Patron	Teacher Win Rate
Hlaalu	85.08%
Duke of Crows	84.79%
Pelin	84.10%
Ansei	83.96%
Rajhin	83.55%
Red Eagle	83.02%
Orgnum	82.03%

Table 7.7: Teacher win rate by patron.

When looking at individual patrons, the win rates are similar, all falling between 82% and 85%. However, specific 4-patron combinations show a larger spread. Out of all 35 possible 4-patron combinations, the highest-performing was Ansei, Duke of Crows, Hlaalu, and Pelin (89.3% over 356 games), while the lowest was Ansei, Orgnum, Rajhin, and Red Eagle (73.7% over 357 games), a 16 percentage point gap.

LT_NN uses random patron selection in all reported experiments. Whether strategic patron selection could improve its win rate is discussed as future work in Section 8.3.

7.8 Cross-Experiment Summary

Table 7.8 summarizes the progression across all experiments.

Exp	Architecture	Parameters	Val Acc	Key Finding
0	V1 MLP	1.77M	55.9%	Severe overfitting
1	V1 + regularization	1.77M	56.3%	Overfitting controlled, +0.4%
2	V1 + 8x data	1.77M	58.9%	+2.6%, plateau at ∼4K games
3b	V1 compact	1.08M	58.8%	Removed unused choice slots
3c	V1 sorted	1.08M	51.6%	Sorting broke position-action link
4	V2 pointer-based	457K	59.2%	Decouples actions from positions
5	V2 + DAgger	457K	59.6%	No consistent win rate gain

Table 7.8: Experiment progression.

The progression from 55.9% to ∼59.5% occurred primarily through data scaling (+2.6%). Neither regularization, architecture changes, nor DAgger moved accuracy beyond the ∼59% plateau, which persisted across five model configurations (84K to 2.3M parameters) and two training methods.

Beyond the accuracy numbers, two qualitative findings shaped the project's direction. Experiment 3c revealed that the positional action encoding is incompatible with reordering input features - a constraint that motivated the V2 pointer-based architecture, which successfully decoupled action scoring from card positions. Experiment 5 (DAgger) showed that collecting training data from learner-visited states does not help when the accuracy ceiling is caused by missing information rather than missing states. The likely limiting factor is the teacher's information advantage from search (Section 4.6), discussed further in Chapter 8.

8 | Conclusions and Future Work

This chapter summarizes the main results, lists the key findings, and outlines directions for future work.

8.1 Summary

This thesis investigated behavioral cloning for learning to play the Scripts of Tribute card game. Through six experiments, a neural network was trained to imitate an expert MCTS bot, progressing from an initial MLP (∼56% accuracy, 1.77M parameters) to a pointer-based architecture (∼59% accuracy, 457K parameters). The final model was deployed as a live bot and evaluated against COG 2025 competition bots, achieving a 36.9% overall win rate. With the setup used in this thesis, behavioral cloning produced a bot that plays competently against heuristic bots (60-100% win rate) but falls short against the search-based bots that dominate the competition (12-15% win rate).

The model achieves ∼59% top-1 accuracy on the validation set (60.41% on the held-out test set) -approximately 15 percentage points above the strongest trivial baseline and with a legal-restricted weighted ROC-AUC score of 0.854, confirming that errors tend toward near-misses rather than random choices. However, this offline accuracy translates to only 12-15% win rate against the MCTS teacher, a gap that illustrates the compounding-error problem: with approximately 9 decisions per turn, even a 59% per-decision success rate produces less than 1% probability of a flawless turn, and each error pushes the game into unfamiliar states where subsequent predictions become less reliable.

The ∼59% accuracy plateau is consistent, persisting across five model configurations, two training methods, and varying data volumes. The most likely explanation is an information gap: the MCTS teacher accesses future states through search, while the student observes only the current visible game state. This suggests the ceiling is caused by missing information in the input rather than insufficient model capacity, as the DAgger results further support, collecting data from learner-visited states did not help when the teacher's decisions depend on information the student cannot observe.

DAgger (Experiment 5, Section 6.2) was investigated to address distribution shift but produced no consistent improvement despite three iterations and 169,000 additional samples. DAgger's formal convergence guarantees do not extend to neural network function approximators [14, 8], though this

theoretical gap alone does not fully explain the result, since DAgger has been applied successfully with neural networks in other domains.

8.2 Main Findings

1. **The ∼59% accuracy plateau is consistent** across five configurations, three model sizes (84K-2.3M), two training methods, and varying data volumes, suggesting the ceiling is set by teacher imitability, not model capacity.
2. **Offline accuracy does not predict live win rate.** Compounding errors over ∼9 decisions per turn reduce ∼59% per-decision accuracy to 12-15% win rate against the teacher.
3. **Positional action encoding constrains the architecture.** Sorting cards lowered accuracy (Experiment 3c), motivating the pointer-based V2 architecture that scores actions by content rather than position.
4. **The model learns meaningful decisions beyond trivial strategies.** The 16 percentage point lift over the strongest baseline is spread across all action types, including types where the trivial baseline achieves 0%.
5. **DAgger did not yield consistent improvement** despite three iterations and 169,000 additional samples, further supporting the information-gap explanation over distribution shift.

8.3 Future Work

Several directions could address the limitations identified in this thesis:

Model-guided search. The trained policy network could be used to guide Monte Carlo Tree Search, following the AlphaZero approach [15] where a policy head provides action priors and a value head evaluates leaf nodes. This would require training a value head capable of position-specific evaluations, likely through self-play [16] or temporal-difference learning rather than behavioral cloning.

Outcome-weighted behavioral cloning. Downweighting decisions from lost games using reward-weighted regression [12, 11] could improve effective teacher quality, though the expected impact is limited given the teacher's 84% win rate.

Patron selection. LT_NN currently selects patrons randomly. The teacher's win rate varies by up to 16 percentage points depending on the patron combination, suggesting that strategic patron selection could improve performance. A small-scale test was inconclusive, and a proper evaluation would require larger sample sizes and logging of the patron draft process.

Richer state representations. Expanding the state encoding to capture more of the available game information, such as card effects or game history, is a potential direction, though whether it would improve accuracy was not tested in this thesis.

Reinforcement learning. Moving beyond imitation to direct optimization of win rate through policy gradient methods, offline RL [1], or sequence-modeling approaches [2] could overcome the teacher imitability bottleneck entirely.

Declaration on the Use of Generative AI

In accordance with Aalborg University's rules on the use of generative AI, the following declaration is made regarding AI tools used during this thesis.
Tool used: Claude (Anthropic), accessed through Claude Code (CLI tool).
How it was used:

- **Idea generation and debugging:** Used to explore design alternatives, debug code issues, and write code documentation and comments. All decisions were made by the author.
- **Data analysis:** Assisted with generating plots, formatting result tables, and verifying numerical claims against source data files. All results were validated by the author.

What was not generated by AI:

- All experimental design decisions, research questions, and methodology choices were made by the author.
- All experiments were executed and results collected by the author.
- The interpretation of results and conclusions are the author's own.

Bibliography

[1] David Brandfonbrener et al. "Offline RL Without Off-Policy Evaluation". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. arXiv:2106.08909. 2021, pp. 4933–4946.

[2] Lili Chen et al. "Decision Transformer: Reinforcement Learning via Sequence Modeling". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. arXiv:2106.01345. 2021.

[3] Rémi Coulom. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search". In: *Computers and Games*. Springer, 2006, pp. 72–83. DOI: 10.1007/978-3-540-75538-8_7.

[4] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[5] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. arXiv:1502.03167. 2015, pp. 448–456.

[6] Levente Kocsis and Csaba Szepesvári. "Bandit Based Monte-Carlo Planning". In: *Proceedings of the 17th European Conference on Machine Learning (ECML)*. Springer, 2006, pp. 282–293. DOI: 10.1007/11871842_29.

[7] Jakub Kowalski et al. "Introducing Tales of Tribute AI Competition". In: *IEEE Conference on Games (CoG)*. IEEE, 2024, pp. 1–8. DOI: 10.1109/CoG60054.2024.10645674.

[8] Jonathan Wilder Lavington, Sharan Vaswani, and Mark Schmidt. "Improved Policy Optimization for Online Imitation Learning". In: *Proceedings of the 1st Conference on Lifelong Learning Agents (CoLLAs)*. arXiv:2208.00088. 2022.

[9] Juho Lee et al. "Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 2019, pp. 3744–3753. URL: https://proceedings.mlr.press/v97/lee19d.html.

[10] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *Proceedings of the 7th International Conference on Learning Representations (ICLR)*. arXiv:1711.05101. 2019.

[11] Xue Bin Peng et al. "Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning". In: *arXiv preprint arXiv:1910.00177* (2019).

[12] Jan Peters and Stefan Schaal. "Reinforcement Learning by Reward-Weighted Regression for Operational Space Control". In: *Proceedings of the 24th International Conference on Machine Learning (ICML)*. 2007, pp. 745–750. DOI: 10.1145/1273496.1273590.

[13] Dean A. Pomerleau. "Efficient Training of Artificial Neural Networks for Autonomous Navigation". In: *Neural Computation* 3.1 (1991), pp. 88–97. DOI: 10.1162/neco.1991.3.1.88.

[14] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2011, pp. 627–635. URL: https://proceedings.mlr.press/v15/ross11a.html.

[15] David Silver et al. "A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through Self-Play". In: *Science* 362.6419 (2018), pp. 1140–1144. DOI: 10.1126/science.aar6404.

[16] David Silver et al. "Mastering the Game of Go without Human Knowledge". In: *Nature* 550 (2017), pp. 354–359. DOI: 10.1038/nature24270.

[17] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: https://jmlr.org/papers/v15/srivastava14a.html.

[18] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. "Pointer Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 28. arXiv:1506.03134. 2015.