Poker Game State Detection

Jack Hung, Luke Moberly, Michael Souliman
Stanford University
450 Jane Stanford Way
{jjhung66, lmoberly, msoul}@stanford.edu

Abstract

We address the challenge of live poker game state detection, creating a (partially) automated poker agent capable of analyzing in-person games and making game-theory optimal moves. Our solution includes card detection, move detection and chip counting, game state tracking, and GTO move suggestion. For card detection, we fine-tuned a YOLOv8 model achieving 99%+ accuracy, and used Grounding SAM with a CNN-based classifier as an alternate method that proved to work better in practice than the YOLO model. Our system processes a live video feed to detect community cards and player moves, feeding this data into a game state tracker that utilizes a GTO solver for optimal move suggestions. For player action recognition, defined action boxes track player moves with GroundingDINO and a linear classifier for chip detection. Our move recommendation system, based on Pavel Tumik's GTO framework, uses extensive simulations to calculate expected values for different moves, guiding our poker agent to make optimal decisions in live games. Future work could include personalizing GTO strategies for individual players based on their action history, though this was beyond our current scope. Our project ultimately integrates vision systems and game theory for real-time poker state detection and automated decision-making.

1. Introduction

We attempt to solve the problem of live poker game state detection. The goal of this project was to create a fully automated poker agent, capable of reading a live game and making a game-theory optimal move. Automated poker bots do exist for online games, but the majority of high-stakes games are still played in casinos. Thus, an automated agent needs to be able to analyze an in-person game. We built an automated system that takes a live stream of a poker table and is able to detect the game state, including what round of betting the game is in, what the moves of other players are, and what the community cards are. Using this system, we are able to make a game-theory optimal move for an automated player in a live poker game.

The problem was broken into five parts: card detection and identification, chip counting, move detection and player attribution, game state tracking, and GTO move suggestion. For card detection and identification, we use a fine-tuned YOLOv8 model on an augmented card dataset. In testing, it was able to achieve 99%+ accuracy, however, in practice it often misread some cards and made the game state incorrect. To mitigate this, we had an alternate method that used Grounded SAM to segment out cards on the table and a CNN-based classifier to identify the cards themselves. This method was slower but worked better in practice. We used GroundingDino and a simple linear classifier to understand the game state of each player for a given round: whether the player folded, bet, and the size of the bet they have made.

To use the system, the action box for each player was defined (the area on the table in which they place their cards during folds or place chips during bets), as well as the area for the community cards (the shared cards during each hand). Frames from the live video feed were parsed into player action boxes and the community card area, which were then fed into their corresponding systems. The community card bounding box was fed into our card detector, which detected and identified any cards present. Frames for each player's action box were fed into GroundingDINO, segmented, and then used the detected objects to infer what move had been made. If there are cards in the action box, the classifier logs a fold, and if there are chips, the classifier will log a bet with its size (based on the chip identification system).

The information from these two vision systems was then fed into our game state tracking system, which utilizes a GTO poker agent to suggest the optimal move when it is the user's turn. Using an existing GTO solver as a framework for our GTO strategy [10], we ran 100 million simulations of poker games with different combinations of hole cards (the cards each player is dealt), community cards, and player bets. The agent uses these simulated hands to calculate the expected value of different moves during each round

of betting. Our hero then chooses the move with the highest expected value. Play continues until all players but one fold or a showdown occurs after the fourth and final round of betting. At this point, our hero simply starts a new game. The game state tracker resets information about hole and community cards, pot size, and player actions, but persists information about each player's stack size.

Future work could include using a player's action history to create individual GTO strategies for each other player at the table. However, because this was primarily a vision project, such work was out of our scope.

2. Related Works

The main task within the vision component of our agent is object detection. For us to create our agent, we need to detect the current state of the game by recognizing not only the cards we currently have, but how much each player has bet along with the move they have made for that round.

Looking at the task of object detection, we use the YOLO architecture [7] for our card recognition system. While other object recognition systems use an R-CNN [2], which uses a selective search algorithm to identify regions to run through a classification CNN, YOLO frames object detection as a regression problem that allows it to run the image through a CNN once, meaning that YOLO works very fast and can run in real-time on a stream of video up to 45 FPS. This works well for our use case as we can continually check frames as we receive them from the stream to check which actions players take and understand what the optimal move to make is in real-time. Additionally, because of the potential to have inaccurate readings of the cards, we sampled 100+ frames and ran the fine-tuned YOLO model of each frame, choosing the classes that were most frequently detected. This increased the accuracy for card identification.

In order to augment our object detection system, we also utilize GroundingDINO [3], which is a transformer-based object detection model capable of identifying and framing specific items from an image based on user prompts. The GroundingDINO model demonstrates great baseline performance in detecting a wide range of objects including poker chips and cards key to our project, however it is not trained to identify specific values of cards and chips. Therefore, we believe that GroundingDINO has its best potential in helping our system quickly lock in on key objects which we can further process in detail.

A related system, GroundedSAM [8], builds off the functionality of GroundingDINO and is capable of directly segmenting the detected object from the background. The model can be used to process detected objects for classification tasks down the line.

3. Methods

3.1. Player Action Recognition

We use a top-down view of the poker table as the input to our system. To process the actions of each player on the board, we use areas of the board that we define as "player action boxes". We set these action boxes during the set-up of our system by drawing bounding boxes around where each player can make their move. This includes adding poker chips to that area if they want to call or raise, or putting their playing cards face down in the area to signify that they have folded. After defining the player action boxes, we then draw a single bounding box around where the community cards are/will be.

To process the current state of a player (their bet/action), we first run Grounding DINO on their action box with the prompt: "a set of poker chips and playing cards on a table". This returns a set of labels of objects from the prompt it has identified along with the coordinates for their bounding boxes. If the object label is "poker chips", we then run a linear classifier on the poker chip to get its color and therefore its value in the bet. If the object label is "playing cards", we assume that the player folded that round. If nothing is detected in the player's action box, we assume that they have not played yet. This limits our state space to a player calling (matching the previous player's bet), folding (ending their round), or raising (betting more than the previous player).

We chose to use a linear classifier for the task of chip color detection because of its speed compared to another type of neural method or using another classification method such as k-NN. We attempted to use a method such as averaging the pixel values in the image and seeing what color it was closest to, but this method did not work well in practice. The linear classifier had the architecture shown in figure 1. We used an SDG optimizer with a learning rate of 1e-3 and momentum of 0.9.

One limitation of this method is that players cannot stack their chips and must lay them out flat, which is a rule the dealer can impose in a standard poker game. One extension of this project for future iterations is that we can take images of stacks of poker chips and train a model to identify the number of chips in the image (potentially looking at the stripe patterns to extrapolate the height of each chip and from there comparing it to the height of the stack to get its value).

3.2. Table Card Recognition

3.2.1 YOLOv8

We chose to use YOLOv8 for card recognition and identification for its speed. The biggest concern we had for card identification was a misread card, as a mistaken read of the community cards would significantly impact the accuracy

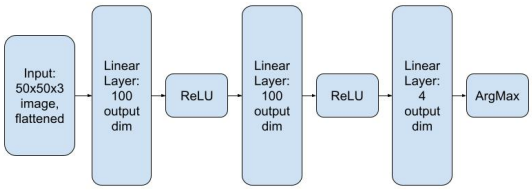

Figure 1: Linear Layer Architecture for Chip Value Recognition

of the GTO agent, rendering it practically useless. As such, we needed an accurate identification model. However, we also wanted to take the running average of card reads over multiple frames. Rather than taking one frame of the table and passing it through GroundingDINO when our game tracking system identified the start of a new round, and using that as the source of truth for the community cards, we wanted to pass dozens of frames through a faster model and take the running average of what it thought were the cards on the table. In this way, even if one frame was poorly read, we had a more accurate read of the cards on the table.

We fine-tuned the YOLOv8 model on a dataset of playing card images overlaid on randomly generated, noisy background images [9]. More information on the dataset is available in section 4.1. We used an AdamW optimizer, 1E-3 learning rate, weight decay = 0.0005, $\beta 1 = 0.937$, $\beta 2 = 0.999$, and 10 epochs.

After a round of betting is complete and the cards are dealt, we take 100 consecutive frames ($\sim$3.5 seconds) and pass the bounding boxes for the community cards section through the fine-tuned YOLO model. YOLO identified the two corners of the cards that have both the rank and the suit, rather than identifying the card as a whole. Based on the number of cards that should be on the table (known by the game state), we take the top predictions from YOLO.

One problem we faced with our method of training on YOLO was the low granularity during actual gametime. During testing, we got near-perfect results on images that were similar in size and resolution to the training images. However, during the game, the camera must be farther away to fit the entire table in frame. As such, the cards in the bounding box are lower resolution and sometimes caused issue with YOLO (as the suit or rank in the corner of the card was sometimes blurry). In particular, 6s and 9s were often confused, as can be seen in image 7. This led us to try an alternative approach.

3.2.2 Grounded SAM + ResNet

An alternative approach we implemented to the YOLOv8 model was to use GroundingDINO to detect and frame community cards, which are then passed to GroundedSAM to segment out the images of the cards. The cards are then passed into a ResNet classifier to identify the type of card (Figure 2).

Here, we used transfer learning on the ResNet 152 model initialized with the ImageNet-1K V2 weights [5] available from PyTorch. We modified the fully connected output layer to output logits corresponding to each of the 52 card classes, then trained the model without freezing layers on the card classification dataset [4] over 18 epochs. We used the SGD optimizer with momentum $= 0.9$ and learning rate $=$ 1E-3.

Unlike with the YOLOv8 approach, we do not average results from multiple frames when using GroundedSAM + ResNet, as this process is much slower. However, we believed that the ResNet could classify playing cards with much higher accuracy in a real-life scenario since GroundedSAM pre-processes images of the detected cards and therefore give the model much more ideal inputs unlike with YOLOv8 alone.

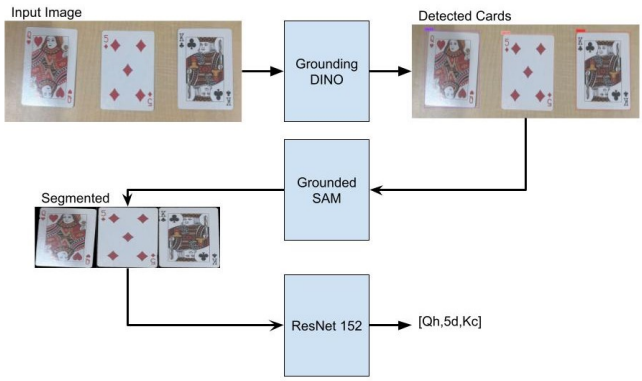

Figure 2: Diagram of GroundedSAM + ResNet Card Detection Pipeline

3.3. Game State Tracking

We initially developed a game state system that allowed for players to check on post-flop rounds of betting by continuously checking the action boxes of all players. If the action classifier had detected a move from player 3, then our game state could infer that players 1 and 2 had checked. However, continuously processing the action boxes of all players, especially in larger 6 or 9-handed games, was too

inefficient. Thus, we assumed for this project that players would either bet or fold each turn, without the option to check. This simplified the game state system immensely, enabling our action classifier to only run on the action box of the player whose turn it is. This system runs until the player with the highest bet is reached again (as the round of betting starts once all players fold or match the highest bet).

3.4. Move Recommendations

Our move recommendation agent was based on work by Pavel Tumik [10]. Pavel created a framework to simulate millions of games based on random combinations of hole cards and community cards. We ran 100 million simulations to create a dataset that included the frequency of specific hands, their success rates across different betting rounds, and the likelihood of it being the winning hand. The poker agent takes these probabilities for any given hand, uses the pot size, player positions, and the possible opponent hand ranges, and calculates the expected value of three possible moves by the hero (fold, call, or min raise). An example from a round is shown below. The hero is on the button in the final round of betting, has pocket aces, is facing a $10 bet from a previous player into a $130 pot. The GTO agent suggests a call.

```
hand cards: [Ah, Ac]
community cards: [7c, Td, 3c, 7d, 8c]
Pot: $130.00, To Call: $10.00
Opponent hand range:
1/9 Pair            :45.9%
2/9 Two Pairs       :37.8%
3/9 Three of a Kind :6.9%
4/9 Straight        :3.0%
5/9 Flush           :3.6%
6/9 Full House      :2.7%
7/9 Four of a Kind  :0.1%

Hand Equity: 90.27%, Type: 2/9 Two Pairs (100.00%)
EV:
CALL  $10.00: +107.35
RAISE $20.00: +97.35
```

Figure 3: GTO Move Suggestion for Hero

There are three shortcomings to this system. First, it doesn't take into account bluffing probabilities. There has been considerable work on determining the likelihood of a move being a bluff using either facial recognition [1] or bet sizing analysis [6], which significantly impacts the expected value of moves our hero can take. We did not consider bluff detection to be in-scope for this project, as statistical analysis of moves would take away from our work on the vision component, and deception detection is an unsolved hard problem. Second, the GTO agent only calculates the expected value of folding, calling, or raising by twice the highest existing bet (the min raise). Other poker strategies suggest doubling the pot in certain situations, raising 3 or 4-fold the highest existing bet, etc... Thus, our poker agent is limited in the types of moves it can suggest, which may cause it to miss out on the truly optimal move. Third, it doesn't update player ranges based on how they play. Players are typically classified as "tight," "loose," or "balanced," which reflects how often they will play a round. For example, a "loose" player is more likely to play bad hands. The GTO system uses a pre-defined range for each player position.

4. Dataset

4.1. Card Detection & Classification

We fine-tuned the pretrained YOLOv8 model with a dataset of playing cards. The dataset [9] is a collection of card images that are placed into a variety of synthetically generated backgrounds. The images contain multiple cards and the bounding boxes of the card corners (containing both the suit and the rank). The images are augmented such that the playing cards are placed at various overlapping angles and rotated.

One shortcoming of the dataset was the specific scale of the cards relative to the image. During live prediction, the model will not be able to accurately detect or identify cards if the cards are too small or large relative to the overall frame/image. This was discussed in section 3.2.1. The predicted labels and confidence of 16 examples are shown in figure 4.

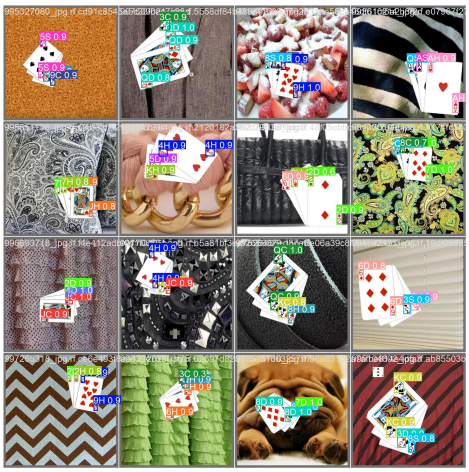

Figure 4: Sample prediction on card dataset

We also performed transfer learning on a pretrained ResNet V1.5 model with a second playing cards dataset. The dataset [4] consists of fully segmented, unobscured playing cards of varying designs and image quality.

Notably, all of the images in the dataset represent front-facing, unobscured playing cards, which means the ResNet

trained on this dataset will likely perform worse if the images captured in real-time are less ideal (Figure 5). However, since we use Grounded SAM to segment the exact image of the card before classifying it with ResNet, we believed that this would not pose a big problem for our experiments.

Figure 5: Cards from the classification dataset, demonstrating different designs and image quality

4.2. Poker Chip Images

For our use case, we not only needed to identify if a player made a bet, but if they did how much they bet. To do so, we trained a linear classifier as described in section 3.1 to categorize the color of the chip. To give each color a value, we have the dealer input the values of each chip when initializing the model when they are also drawing the bounding boxes around each player's action space.

To generate the dataset we used to train the linear classifier for poker chip color detection, we used Grounding Dino on a live feed of the poker table to extract images of just the poker chips. Once we had images of just the poker chips, we built a simple UI used to label the images into one of the four color classes and hand-labeled the image colors, along with downsampling the resolution to 50 pixels by 50 pixels to limit the number of parameters learned with the first linear layer. To attribute value to poker chips identified in a player's action space, we used the color of the poker chip as that was the most defining feature of the poker chip that distinguished it from other chips. You can see an example of what these images look like in figure 6.

One limitation of this method is that we only had images of one set of poker chips from this method (as we only had one poker set at the time of dataset generation). This means that our classifier performs poorly on other sets of poker chips, which may have a different design that our classifier hasn't learned. To mitigate this, we could repeat

Figure 6: Sample images from poker chip dataset

the process above with several different sets of poker chips (which would also give us a large class space as we could deal with more colors). We could also scrape images online for poker chips that would help diversify our dataset, but for our use case in this project our method worked well. Although our classifier was overfit on our set of poker chips, it achieves 100% test accuracy on pictures of poker chips from our set that it has not seen before so it still works well for our project's use case.

5. Results & Discussion

5.1. Demo

You can find a video of our end-to-end system running here. A transcript of what is happening in the video can be found in the appendix.

5.2. Card Detection

5.2.1 YOLOv8

The Base YOLO model did fairly well detecting the bounding box of playing cards (the box loss wasn't too significantly improved upon by the fine-tuned model, at least in comparison to the classification loss), but did poorly at classifying the suit and rank of the cards. The fine-tuned model, trained for 10 epochs with an AdamW optimizer, significantly improved the card identification capabilities of the model. This is shown in Table 1. The confusion matrix for each of the 52 playing cards is shown in Figure 7.

	Box Loss	CLS Loss	Recall	Precision
Base YOLO	1.4712	3.8849	0.28579	0.56042
Fine-Tuned	1.1535	0.65933	0.99656	0.99599

Table 1: Base vs Fine-Tuned YOLOv8 Model

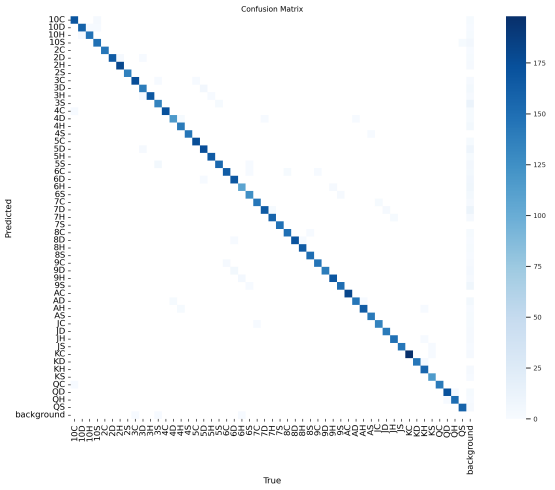

Figure 7: Confusion Matrix for True vs Predicted Card Identifications

5.2.2 ResNet Card Classification

The base ResNet V1.5 model had poor performance in card classification, achieving 8.28% baseline test accuracy on a generalized dataset of many cards with different designs. However, the transfer learned ResNeT V1.5 model trained for 18 epochs with stochastic gradient descent achieved a classification test accuracy of 80.67% on the same dataset. From the training graph, it is clear that further optimizations can be made through hyperparameter tuning to decrease the overfitting which occurs late in the training (Figure 8). We decided the results here were sufficient for our use case since the ResNet was trained on a dataset with much more varied and sometimes convoluted card designs and image quality, and therefore our model would be expressive enough in practice (Figure 9).

Additionally, while we do not have a quantitative score, initial experimentation has demonstrated that the ResNet model, which is trained on images of unobscured cards, is incapable of classifying cards that are partially covered (Figure 10), a key flaw compared to the YOLO model.

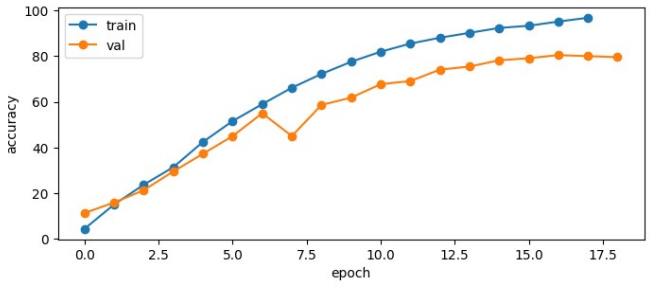

Figure 8: Training and Validation Classification Accuracies Over Time

Figure 9: Example of a Four of Diamonds design in the test set that is more difficult to qualify

Figure 10: Segmented, partially obscured Jack of Spades, which the classifier identifies as a Two of Spades

5.3. Player Action Detection

5.3.1 Grounding Dino for Segmentation

Looking at the performance of Grounding Dino qualitatively, we saw good performance at segmenting out the poker chips and cards in the image under ideal circumstances. Examples of what GroundingDino was able to segment out are shown in figure 11. As seen in the image, all poker chips and cards were able properly segmented, allowing us to correctly update the action for each player.

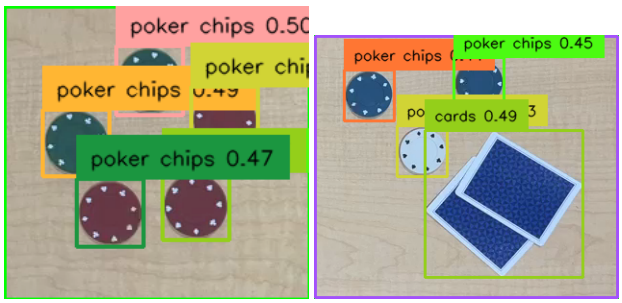

Figure 11: Grounding Dino run on images of poker chips and playing cards

Grounding Dino performed well for our use case given some prompt engineering and fine-tuning for thresholds, but still had some notable failure cases. One of these failure cases was having the poker chips touch each other, which

would cause Grounding Dino to segment them all into one image and not individual chips. Another notable failure case was Grounding Dino not segmenting out all of the poker chips, but we learned that this was mainly caused by the player action boxes being drawn too large, meaning that the poker chips took up less of the image which made them harder to segment.

5.3.2 Chip Identification

Despite the limitations in our dataset mentioned in section 4.2, we were able to achieve 100% accuracy on the test set with our linear model. Training and validation metrics for each epoch can be found in figure 12.

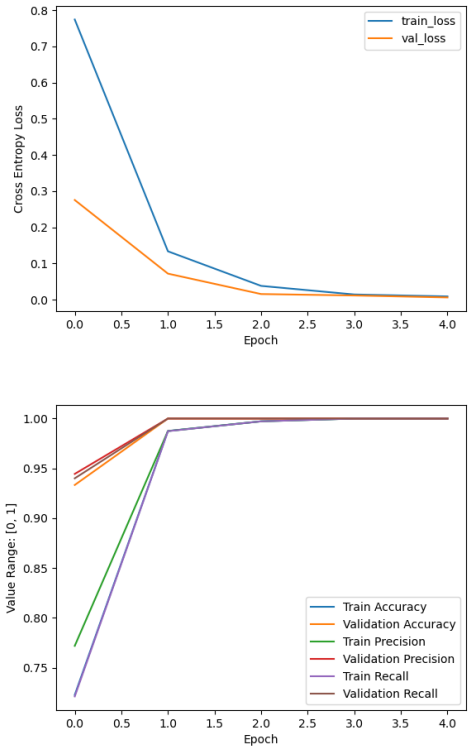

Figure 12: Loss and other metrics when training the chip linear classifier

6. Conclusion

By synthesizing the outputs of several systems including card detection, move detection, player attribution, game state tracking, and a GTO player, we have built a partially automated game state detection system for poker that is able to automate decision-making for a player. Utilizing transfer learning, we fine-tuned a YOLOv8 model for card detection, which did not work the best in practice so we pivoted to a pipeline that used Grounding SAM to segment the cards from the image and a fine-tuned ResNet to classify them. We also used GroundingDINO with a linear classifier for player action recognition. This made it such that we were able to minimize latency when processing the camera input and updating our game state.

The highest-performing algorithms in our system were the Grounding SAM paired with a ResNet for card detection, and GroundingDINO paired with the linear classifier for mode detection. The YOLOv8 model demonstrated high accuracy (and more importantly speed) in card detection on the dataset, but in practice performed worse than the Grounding SAM method.

Looking forward, several areas could be explored to further enhance our system. Incorporating individual player strategies by personalizing GTO suggestions based on historical data could significantly improve decision-making. Additionally, integrating bluff detection using facial recognition or bet sizing analysis would add a deeper level of strategy and realism. Expanding the training dataset to include diverse images of different poker chips and cards would improve the system's adaptability and robustness. These improvements would refine decision-making and increase the system's applicability, building on the solid foundation established by our current work.

References

[1] J. Feinland, J. Barkovitch, D. Lee, A. Kaforey, and U. A. Ciftci. Poker bluff detection dataset based on facial analysis. In *International conference on image analysis and processing*, pages 400–410. Springer, 2022. 4

[2] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. 2

[3] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, J. Zhu, and L. Zhang. Grounding dino: Marrying dino with grounded pre-training for open-set object detection, 2023. 2

[4] G. Piosenka. Cards image dataset-classification. https://www.kaggle.com/datasets/gpiosenka/cards-image-datasetclassification/data. visited on 2024-06-3. 3, 4

[5] PyTorch. ImageNet V2 Pre-trained Weights for ResNet-152. https://pytorch.org/vision/stable/models.html#torchvision.models.resnet152, 2022. 3

[6] R. Ranca. Identifying features for bluff detection in no-limit texas hold'em. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013. 4

[7] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. 2

[8] T. Ren, S. Liu, A. Zeng, J. Lin, K. Li, H. Cao, J. Chen, X. Huang, Y. Chen, F. Yan, Z. Zeng, H. Zhang, F. Li, J. Yang, H. Li, Q. Jiang, and L. Zhang. Grounded sam: Assembling open-world models for diverse visual tasks, 2024. 2

[9] A. Startups. Playing cards dataset. https://universe.roboflow.com/augmented-startups/playing-cards-ow27d, aug 2023. visited on 2024-05-16. 3, 4

[10] P. Tumik. Poker ml. 1, 4

7. Appendix

7.1. Demo Transcription

In the demo video, we start by saying where the automated agent is in the game (placing the hero on them, meaning they have the big blind and player 0 is the first to make a move). We then draw the bounding boxes around each player (pressing d to delete the previous bounding box) and the community cards. We then input the cards the player has manually, which were the 2 of hearts and 10 of hearts.

We then wait for player 0 to make their move, which is a bet of 2 green chips, or $2, and then wait for player 1 to make their move, which is a raise of 2 blue chips, or $10. We then placed the community cards down and waited for the card detector to read them, which gave us that the community cards were the 2 of spades, 10 of spades, and 7 of diamonds.

It was then our agent's move, which it recommended that betting $1 had the highest expected value, so that was the move that it made. It then repeated the process of waiting for players 0 and 1 to make a move, where player 0 raised to 2 blue chips, or $10, and player 1 raised to 2 red chips, or $20. From there, it was the agent's move again where the GTO player said the move with the highest EV was to call the last bet for $20, so that is what our player did.

We then placed the next community card, which was the 10 of diamonds. It was then player 0's move, where they bet 2 red chips, or $20, and then player 1's move, where they folded and placed their cards in their action box. This made it so that their card_bool was now true to signify that they folded.

From there, our GTO player suggested that calling player 0's bet was the best move, so we called their $20 bet. Then the last community card was shown, which was the 10 of clubs, giving our agent 4 of a kind and winning.

8. Contributions

Luke fine-tuned the YOLOv8 model. He also wrote the scripts used to track the game state and run the GTO player. He also worked on the engineering infrastructure used to process the video feed from OpenCV.

Jack worked on card detection using Grounded SAM. He also finetuned the ResNet classifier used in that model, along with contributing to the game state tracking script.

Michael worked on move detection, including using Grounding Dino to segment the player action boxes. He also trained the linear classifier used for poker chip identification. He contributed the boilerplate code to process the images with OpenCV.

All GitHub repos used are included in our references section.