A Appendix

A.1 Prompting details

System prompt 1: *"You are an expert Hanabi player"*

System prompt 2: *"You are an expert Hanabi player focused on maximizing team coordination and achieving high scores with minimal mistakes. Follow these principles: Efficient Clue-Giving: Provide clues that give maximum information, using finesse and double clues to benefit multiple players. Deduction: Track played/discarded cards and deduce your own cards based on clues and game state. Avoid discarding critical cards. Disciplined Play: Play and discard safely, minimizing risk while optimizing the team's progress. Team Coordination: Follow team conventions and use subtle cues (timing, actions) to communicate intent without verbal clues. Score Maximization: Manage clue tokens and pace the game to ensure enough clues for critical moments."*

A.2 Dataset details

The dataset is acquired through self-play mode, utilizing a pre-trained OBL agent in the Hanabi game. Trajectories are filtered selectively with a gameplay score exceeding 20. Then, these trajectories are broken down into state-action pairs to suit language model training. During the initial data exploration, we found the action categories are imbalanced as shown in 7, hence the language model overfits to discard 4 based on the confusion matrix for the prediction. To avoid that, we did categorical sampling consisting of 2200 samples per action type, aggregating to $44,000$ instances. Then we checked for duplicate states and dropped them, there were approximately 100 duplicates as this could mislead the model's learning. After which, 10% of the dataset is reserved for testing by random sampling. Further, the dataset is split into 90% for train and 10% for validation.

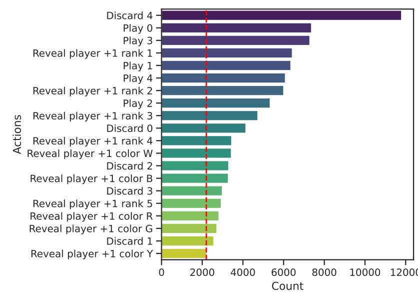


Figure 7: *Visualizing the number of actions available in the dataset to create a diverse dataset of Hanabi gameplay in the form of text.*

A.3 How Good LLMs are in playing Hanabi?

To adapt the LLaMA to the gameplay, we use Low-Rank Adaptation, or LoRA (Hu et al., 2021a), which learns a low-rank decomposition matrices into each layer of the transformer architecture and freezes the pre-trained model weights. Thereby, significantly reducing the trainable parameters. We conducted fine-tuning experiments with LLaMA-7B weights with classifier using varying data sizes $[200, 500, 1000]$ and LoRA ranks $[32, 64, 128]$ for 10 epoch. Despite these parameter variations, the gameplay scores remained suboptimal level of around one as shown in 8. This highlights the challenges in achieving effective gameplay performance for current large languge model on playing hanabi.

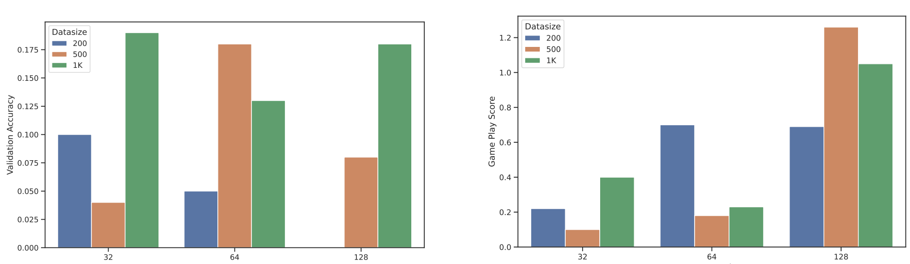


Figure 8: *Evaluation of Low-Rank Adaptation (LoRA) in LLaMA-7B finetuning, showcasing the impact on a) Validation Accuracy and b) Game Play Score. The experiments involve varying data sizes* $[200, 500, 1000]$ *and LoRA ranks* $[32, 64, 128]$.

A.4 Ablation studies

A.4.1 The role of scaling the dataset and different model variants

The dataset size emerges as a pivotal factor influencing gameplay scores. As the amount of training data increases there is a gradual increase in validation and the gameplay score. When the training percentage is equal to or less than 10% the games scores were poor ranging around 1 out of 25. In contrast, the gameplay score sharply increases when using 25% of the data as shown in 9b. Nevertheless, the performance plateaus at a game play score of approximately 9 for both 75% and 100% , indicative of reaching a saturation point, affirming the sufficiency of the dataset size for effective model training.

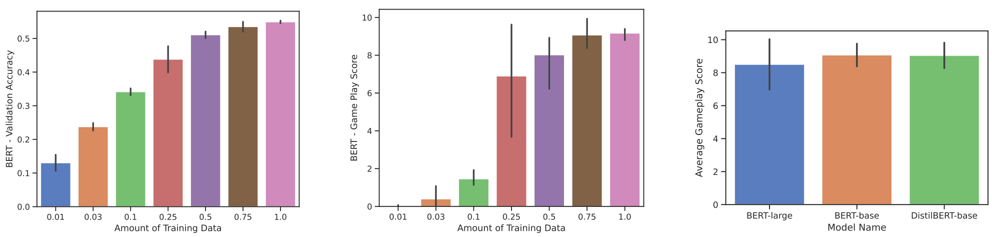


Figure 9: *Analysis of the impact of training data amount on BERT, examining a) BERT Validation Accuracy, b) BERT Game Play Score across different percentages of training data, and c) BERT model variants with varying parameter sizes.*

In our experimentation, we varied the model parameter sizes—ranging from DistilBERT with $66M$ parameters to BERT-base-uncased with $110M$ parameters and BERT-large-uncased with $340M$ parameters. We observed that DistilBERT achieves a competitive gameplay score of approximately 8.7 after 600 game runs 9c. On top of the performance considering the fast inference and low memory usage, DistilBERT was chosen as a candidate for integration with reinforcement learning through distillation.

A.4.2 The role of discard information

We examined the impact of incorporating the discard pile into the observation. Surprisingly, we discovered that utilizing the discard pile did not contribute to any improvement in game scores as show in the Figure 10. Rather, it resulted in a doubling of the sequence length of the language model. Given the need for fast inference in the reinforcement learning pipeline, we opted to exclude discard pile information from the observation during both language model training and inference. Nonetheless, there is a potential for heuristic-based approaches, to explore the idea of creating derived information from from the discard pile, potentially leading to a more concise sequence length and better game score.

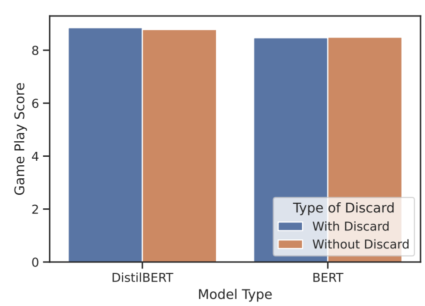
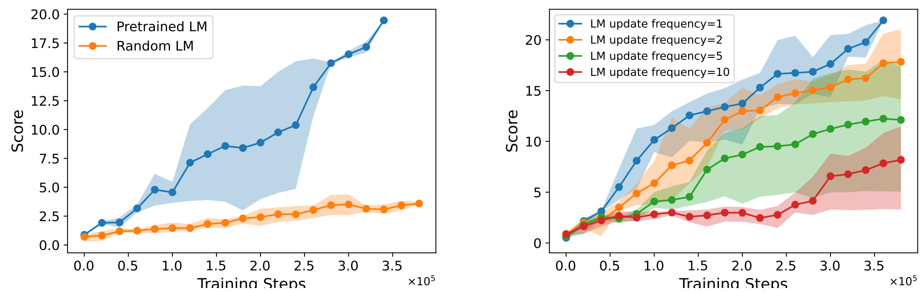


Figure 10: *Evaluation of the discard pile's role in the game is assessed by comparing game scores with the presence and absence of the discard pile in the observation during training.*

A.5 Training details

A.5.1 LM initialization and update frequency

We train two R3D2 agents in a 2-player Hanabi setting: one using a pre-trained language model (LM) and the other with the same architecture but randomly initialized LM weights. Figure 11a shows that learning from pre-trained weights significantly improves the sample efficiency. Additionally, we test updating the LM less frequently with periods of 1, 2, 5, and 10 training steps per LM update to examine whether the original pre-trained weights provide sufficient representations for playing Hanabi or if fine-tuning is necessary. Our results, presented in Figure 11b, indicate that updating the LM parameters is essential for effective learning.



(a) Pretrained vs random LM weights

(b) Frequency of updating the LM

Figure 11: Impact of pre-trained weights and update frequency on learning efficiency. (a) Performance difference between R3D2 agents trained with pre-trained language model (LM) weights versus randomly initialized LM weights, showing significant improvements in sample efficiency with pre-trained weights. (b) The effect of varying the frequency of LM updates, highlighting that frequent updates are critical for effective learning in the Hanabi environment.

A.5.2 R3D2 TRAINING SETUP

A.5.3 LANGUAGE MODEL SETUP

The model's finetuning process begins with a set of training instances, denoted as (S, A) drawn from the dataset $\mathbb{D}$ where $S \in \{s_0, s_1, .., s_n\}$ and $A \in \{a_0, a_1, .., a_n\}$. Within this set, s and a represent a state and its corresponding noisy labelled action, respectively, and n represents the number of examples in the dataset. The training objective of BERT, DistilBERT, GPT2-Classifier is,

$$L_{CCE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} a_{ij}\log(\hat{a}_{ij}) \tag{1}$$

Where N is the batch size. C is the number of classes. a_{ij} is the true probability of class j for the i-th example in the batch and $\hat{a}_{ij}$ is the predicted probability of class j for the i-th example in the batch.

The training objective of GPT-2 Generative is to minimize the cross-entropy loss, denoted as $\mathcal{L}$, and do the finetuning of the model. The cross-entropy loss is mathematically defined as follows:

$$\mathcal{L}_{LLM} = -\mathbb{E}_{(S,A)\sim D}\log p(A|S) \tag{2}$$

Where $p(S|A)$ represents the conditional probability of predicting an action A, given the state S. The goal is to optimize these parameters, by minimizing the cross-entropy loss. We finetune the model to generate responses that better align with Hanabi game. The learning graph of validation accuracy with the game play score for each epoch is logged to understand the trend in the Figure 12(a,b). Mostly the Validation score and game score is getting saturated at around 4th epoch.

A.5.4 SOFTWARE DETAILS

The code was implemented using PyTorch, and pre-trained language models were loaded using Huggingface. To gain insights for this paper, we employed Weights & Biases (Biewald, 2020) for experiment tracking and visualizations. Lastly, plots are created using the seaborn package. For RL algorithms, we used OBL agent (Hu et al., 2021c) to collect the expert trajectory and RL Hive (**?**) to train the algorithm.

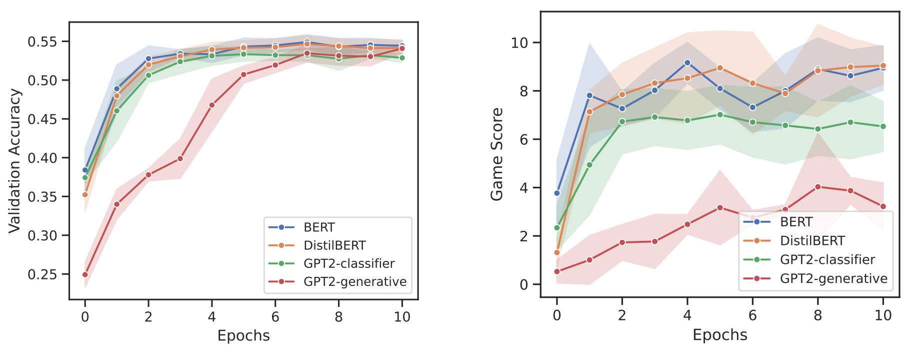


Figure 12: Learning graph for (a) Validation accuracy plotted against(b) Game play score, for each epoch for different language model providing insights into the observed trends during the training process.