Language-Guided Value with Latent Dynamics for Textual Card Game Agent

Wannian Xia[1,2], Shuang Xu[2], Bo Xu[1,2]
[1]School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China
[2]Institute of Automation, Chinese Academy of Sciences, Beijing, China

***Abstract*—Textual card games, such as Hearthstone, offer a rich environment for exploring decision-making integrated with language understanding, yet achieving efficient policy learning across various game scenarios remains a significant challenge. In this paper, we tackle this issue by introducing a new framework that integrates large language models (LLMs) with reinforcement learning (RL) agents to improve training efficiency. Our approach leverages a fine-tuned T5 model to encode and interpret card strategies expressed in natural language, yielding a language-guided value function that computes Q-values with language embeddings. We further augment self-play RL training of the agent with an auxiliary transition loss in the latent space. The agent learns from scratch to capture the complex dynamics of the game environment, facilitating efficient policy learning across a large set of cards and decks. Experimental results demonstrate that our method significantly enhances learning efficiency, enabling the agent to effectively learn from a massive dataset of over 5,000 different decks and to generalize on novel cards. These findings highlight the promise of incorporating pretrained LLMs in RL agents for complex decision-making problems.**

***Index Terms*—Large Language Models, Reinforcement Learning, Game Playing, Neural Networks, Latent Space**

I. Introduction

Language plays a fundamental role in human cognition and intelligence [1]–[3], yet integrating robust language understanding into reinforcement-learning agents remains a key challenge for AI systems. Textual card games—where game rules and effects are conveyed through rich natural-language descriptions—offer an ideal benchmark for this integration [4]–[7]. One prominent example is Hearthstone, as it features thousands of cards whose effects encode conditional triggers and synergies, creating an enormous strategy space under hidden information and stochastic outcomes. Training a Hearthstone agent therefore demands capturing the rich semantic information encoded in each card and leveraging this understanding during RL to guide effective decision-making.

Previous work has explored how to represent the many distinct cards in Hearthstone. A straightforward solution is to use one-hot embeddings [8], but this approach suffers from extreme sparsity and is unable to encode any semantic similarity initially. Xia et al. [9] applied an off-the-shelf pretrained language model to embed card text. However, their experiments were confined to a small dataset of cards, and were not evaluated on larger, more diverse game scenarios. In this work, we introduce a Hearthstone meta game in which each round's deck is selected at random. We assemble a large-scale dataset of decks, thereby substantially increasing both the meta game's stochasticity and the diversity of game scenarios. This expanded variability poses a significant challenge to the learning efficiency of the agent. Implementing methods like counterfactual regret minimization (CFR) [10], [11], which is effective in partially observable poker games like Texas Hold'em [12], [13], can be problematic as the branching factor and number of information sets explode to intractable levels with thousands of game cards and decks, and the game abstraction may collapse key text-driven distinctions.

In our work, to tackle these challenges, we propose to condition the value head of our RL agent on the semantic embedding produced by a fine-tuned LLM, and require the agent to learn the latent dynamics of observations during self-play RL training. Concretely, we first adopt supervised fine-tuning (SFT) on the pretrained T5 model to generate the usage tip of a card given its effect description, yielding a compact embedding from the encoder to capture rich, domain-specific semantics. With a total of 6,891 cards available in the environment, we use GPT-4o to generate the usage tips of each card to avoid the need for manual data annotation. Then, each card's natural-language description is passed through the fine-tuned T5 encoder to produce a dense vector, which is concatenated with the agent's other observed features before Q-value regression. Our experiments demonstrate that this language-guided value network can exploit semantic similarities between cards to bootstrap value estimates and dramatically reduce sample complexity across thousands of distinct card interactions.

Inspired by Dreamer [14]–[17], we further tackle the training difficulty induced by the combinatorial explosion of deck configurations, which results in diverse game scenarios. To address this, we introduce an auxiliary loss that forces the agent to predict its next latent representation given the current observation and action. Unlike Dreamer, we do not require the agent to perform a full game rollout on the imagined latent space, thereby avoiding the biases of a full world model rollout in the partially observable environment. We train our agent on a large corpus of over 5,000 unique decks collected from the Internet, demonstrating that our method improves learning efficiency when exposed to vast and diverse game scenarios.

To summarize, our contributions are as follows: 1) We introduce a highly efficient training pipeline and model framework for the challenging zero-sum textual card game Hearthstone.

Corresponding Author: xubo@ia.ac.cn. This work is supported by the STI 2030 - Major Projects (No. 2021ZD0201500).

With the combination of LLM-based card embeddings and latent-dynamics supervision in self-play RL, we dramatically speed up training convergence and improve strategic play of the agent; 2) We release our entire implementation[1] along with a curated dataset of over 5,000 unique Hearthstone decks to enable reproducibility and drive further research.

II. Approach

In this section, we first introduce the formalization of our problem, and describe how we design the dataset and perform fine-tuning on a T5 model in Section II-B. We also illustrate how we incorporate this fine-tuned T5 model into the design of our agent. Then, in Section II-C, we describe how we train the agent with an auxiliary transition loss during self-play RL.

A. Formalization

We formalize the game as a partially-observable Markov decision process, and introduce our notation below. We define the set of observations as $\mathcal{O}$ and the set of actions as $\mathcal{A}$. At each time step t, the agent receives an observation $o_t \in \mathcal{O}$, performs an action $a_t \in \mathcal{A}$, and transitions to the next observation o_{t+1}. The agent's goal is to learn a policy $\pi(a_t|o_t)$ that maximizes the expected return $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$. In our setting, the environment only provides a sparse reward r_t, where the agent receives a reward of 1 only upon winning a game and a reward of -1 upon losing. Intermediate heuristic rewards are not provided during the game.

B. Fine-Tuning T5 with Game Strategy Dataset

Given the game dynamics described in natural language, the agent should effectively capture the characteristics embedded within these textual descriptions. Although it is possible to use off-the-shelf embedding models on the card descriptions, the embeddings they produce may fail to capture strategic nuances of the game. For example, two cards with similar text descriptions will yield similar embeddings from a pretrained model without being fine-tuned, yet their actual strategic uses can differ dramatically in Hearthstone, as illustrated by the example in Fig. 1. As shown in Fig. 1, *Deal 2 damage to all enemy minions* and *Deal 3 damage to ALL characters* have similar surface text but vastly different strategic implications. To capture such nuances, we design a fine-tuning task for an LLM to generate strategy-aware embeddings. We also include a deck strategy dataset so the fine-tuned model can extract deck-level strategies as a reference for the agent. We employ the pretrained T5-base model and fine-tune it in a text-to-text manner consistent with its pretraining paradigm. For the card dataset, the input includes the card's name and description. The output label is the corresponding strategy of that card described in natural language. For the deck dataset, we collect 5,032 manually designed decks from the Internet[2]. These decks also contribute to the complexity of the meta game. To construct a text-to-text fine-tuning task on these decks, we use the description of the deck strategy as input, and its content described concisely in natural language as output.

To obtain strategy labels for over 6,000 cards and 5,000 decks, we use GPT-4o to generate synthetic data. We describe the complete content of each deck in natural language. This description encompasses detailed information such as the names and descriptions of all the cards in the deck, their mana costs, attack values, health points, and other pertinent attributes. We then utilize these comprehensive descriptions as prompts for the GPT-4o model, instructing it to generate strategies for each deck and card. After fine-tuning, we obtain vector representations for each card and deck using only the encoder of the T5 model. We then apply mean pooling over the encoded states. This results in a single embedding vector for each card or deck. To ensure consistency across embeddings, we normalize these vectors to have a unit L2 norm. Each input is thus represented as a 768-dimensional normalized vector, providing the agent with semantic information derived from textual descriptions.

C. Self-Play RL Training with Latent Dynamics

TABLE I: Architectural settings. All three modules share the same transformer encoder configuration except for the output dimension.

Setting	**Value**
Shared Transformer Encoder	
Input / Hidden Dimension	272
Number of Layers	4
Number of Attention Heads	8
Feed-forward Dimension	256
Dropout	0.1
Module-Specific Output Dimension	
Representation / Latent Dynamics	272
Policy	1
Linear Embedding (Representation Module)	
Hand Card Features	$23 \rightarrow 16$
Minion Features	$26 \rightarrow 16$
Hero Features	$31 \rightarrow 16$
Language Embedding (Representation & Policy)	
T5 Output Projection	$768 \rightarrow 256$

In Hearthstone, part of the state transitions is described in natural language. For the transition from (o_t, a_t) to o_{t+1}, we aim to generate representations with o_t and a_t, so that the agent can reason about these transitions for latent dynamics. With a pretrained and fine-tuned language model, this reasoning is expected to facilitate policy training across various game scenarios. To do this, we introduce an additional auxiliary loss in the latent space of the agent's inference. Specifically, we segment the agent's model into three components: a representation module f_θ which encodes the original observations and actions: $h_t = f_\theta(o_t, a_t)$; a policy module q_ϕ which approximates the action values: $Q(o_t, a_t) = q_\phi(h_t)$; and a latent dynamics module w_ψ which performs reasoning over latent space transition dynamics: $\hat{h}_{t+1} = w_\psi(h_t)$. The

[1] https://github.com/WannianXia/GeneralizableHS

[2] All decks are collected from hearthstonetopdecks.com.

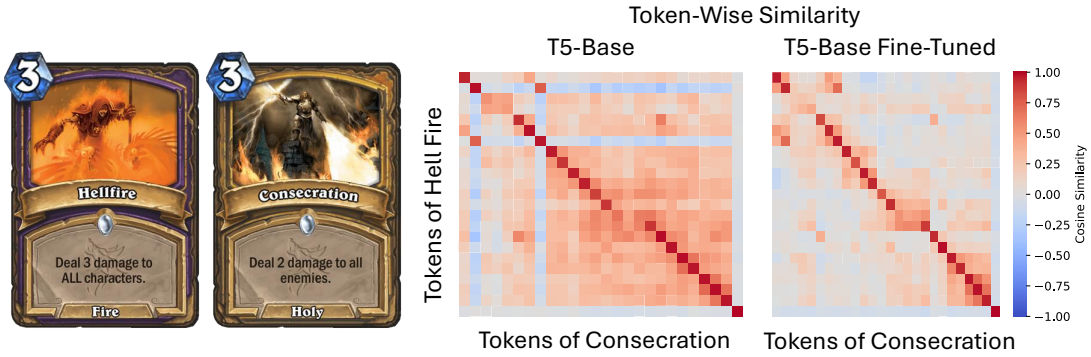


Fig. 1: Two example cards *Hellfire* and *Consecration* in Hearthstone, and a comparison of their token-wise similarity in embedding space. With our fine-tuned T5, the cosine similarity decreased from 0.4198 to 0.3056, even though their descriptions are similar.

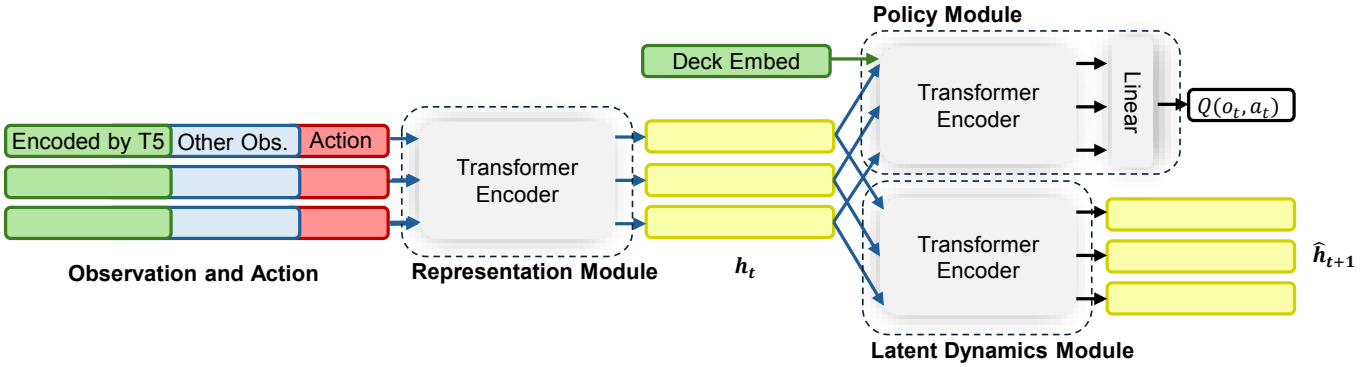


Fig. 2: Our model consists of three transformer encoders for different functionalities: a representation module, a policy module, and a latent dynamics module. The embedding generated by a fine-tuned T5 encoder is concatenated with other observations and the action of the agent, which are then encoded to a latent space by the representation module. The policy module takes the latent representation and the deck embedding to estimate the Q-value, while the latent dynamics module constructs the latent representation of the consecutive observation.

structure of our model is shown in Fig. 2. Each of the three modules is implemented as a 4-layer transformer encoder, constituting a deep reinforcement learning architecture. For each entity in the observation (hand cards, minions, or heroes), we use a linear layer to project their raw features into a 16-dimensional vector, which is then concatenated with a 256-dimensional language embedding obtained by projecting the 768-dimensional T5 output. This yields a 272-dimensional input token for each entity. The detailed architectural settings are summarized in Table I.

We adopt the Actor-Learner architecture [18], [19] to train all three modules simultaneously. The agent interacts with the environment to generate trajectories $\tau = \{(o_t, a_t, o_{t+1})\}_{t=0}^{T}$ in each actor. Upon the completion of each game (trajectory), we compute the discounted return G_t at each time step and store these data pairs in a buffer $\mathcal{D}$ shared with the learner process. The learner constantly samples batches from $\mathcal{D}$ and updates its parameters.

The policy module is trained to minimize the mean squared error (MSE) between the predicted Q-values and the discounted returns from the buffer:

$$\mathcal{L}_{\text{policy}}(\theta, \phi) = \mathbb{E}_{(o_t, a_t, G_t)\sim\mathcal{D}} \left[\left(q_\phi(f_\theta(o_t, a_t)) - G_t \right)^2 \right]. \quad (1)$$

The target for the latent dynamics module is the next hidden state $\bar{h}_{t+1}$ obtained by encoding the next observation o_{t+1} using the representation module with a stop-gradient: $\bar{h}_{t+1} = \text{sg}(f_\theta(o_{t+1}, \text{mask}(a_{t+1})))$, where $\text{sg}(\cdot)$ denotes the stop-gradient operator. Note that the action a_{t+1} is masked here. The stop-gradient ensures that only the dynamics module w_ψ and the prediction pathway through f_θ are updated by the dynamics loss, preventing representation collapse where both the prediction and target converge to a trivial constant. We apply a softmax with temperature $\tau = 0.1$ to both the predicted and target hidden states to convert them into probability distributions, and train the latent dynamics module by minimizing the cross-entropy between them:

$$\mathcal{L}_{\text{dynamics}}(\theta, \psi) = \text{CrossEntropy}\left(w_\psi(f_\theta(o_t, a_t)),\ \text{sg}(\bar{h}_{t+1}) \right) \quad (2)$$

The total loss is then a combination:

$$\mathcal{L}_{\text{total}}(\theta, \phi, \psi) = \mathcal{L}_{\text{policy}}(\theta, \phi) + \beta \mathcal{L}_{\text{dynamics}}(\theta, \psi), \quad (3)$$

where β is a hyperparameter that balances the two loss terms. The learning algorithm is shown in Algorithm 1. We optimize all parameters using RMSProp with a learning rate of 0.0001 and gradient clipping at a maximum norm of 40.0 to stabilize training. The smoothing constant α of RMSProp is set to 0.99, and ϵ is set to 10^{-5} for numerical stability. The agent uses an ϵ-greedy exploration strategy with $\epsilon = 0.01$. The discount factor γ is set to 1.0 since the game has finite-length episodes and we use undiscounted Monte-Carlo returns. The complete training hyperparameters are shown in Table II.

Algorithm 1 Monte-Carlo Sampling with Latent Dynamics

Initialize global model parameters θ, ϕ, ψ, hyperparameters β
Initialize replay buffer $\mathcal{D}$
while True **do**
  Sample a mini-batch of transitions (o_t, a_t, o_{t+1}, G_t) from $\mathcal{D}$
  Compute hidden state: $h_t = f_\theta(o_t, a_t)$
  Predict Q-value: $Q_{\text{pred}} = q_\phi(h_t)$, next hidden state: $\hat{h}_{t+1} = w_\psi(h_t)$
  Compute target hidden state with stop-gradient: $\bar{h}_{t+1} = \text{sg}(f_\theta(o_{t+1}, \text{mask}(a_{t+1})))$
  Compute latent dynamics loss: $L_{\text{dynamics}} = \text{CrossEntropy}\left(w_\psi(f_\theta(o_t, a_t)), \text{sg}(\bar{h}_{t+1})\right)$
  Compute policy loss: $L_{\text{policy}} = (Q_{\text{pred}} - G_t)^2$
  Update parameters θ, ϕ, ψ to minimize $L_{\text{total}} = L_{\text{policy}} + \beta L_{\text{dynamics}}$
  Broadcast updated parameters θ, ϕ, ψ to all actors
end while

TABLE II: Training hyperparameters for our agent

Parameter	Value	Description
Training Settings		
Batch Size	32	Learner batch size
Learning Rate	0.0001	Base learning rate
α	0.99	RMSProp smoothing constant
Momentum	0	RMSProp momentum
ϵ	1×10^{-5}	RMSProp epsilon
Max Grad Norm	40.0	Maximum norm of gradients
RL Parameters		
γ	1.0	Discount factor
ϵ	0.01	Exploration probability
β	0.5	Latent dynamics training loss weight

III. Experiments and Results

In this section, we aim to demonstrate that our method can outperform other methods on this challenging meta game, where during self-play RL the agent's deck is randomized in every game. We also conduct ablation studies to analyze the contribution of each component of our method.

TABLE III: Head-to-head win rates on random mirror matches. Each cell shows the win rate of the row agent against the column agent. Results are averaged over 10,000 games with standard deviation computed across 10 batches of 1,000 games, except for GPT-5 mini which is evaluated on 500 games.

	Our Agent	Cardsformer	GPT-5 mini	T5 Agent	Random
Our Agent	–	**60.3%**±1.2	**84.6%**	**93.8%**±0.9	**99.4%**±0.3
Cardsformer	39.7%±1.2	–	71.2%	85.4%±1.4	97.2%±0.6
GPT-5 mini	15.4%	28.8%	–	68.4%	86.8%
T5 Agent	6.2%±0.9	14.6%±1.4	31.6%	–	74.6%±1.8
Random	0.6%±0.3	2.8%±0.6	13.2%	25.4%±1.8	–

A. Performance on Random Training Decks

We compare our agent with several other models, including an LLM agent based on GPT-5 mini, a T5 agent without fine-tuning, and a random agent. For the GPT-5 mini agent, we provide a comprehensive language description of the game and a list of all available actions for the agent to choose from. For the T5 agent, we compute the conditional probability $P(Y \mid X)$ of the model generating the target action text Y given the state description X, and we select the action with the highest probability to perform. Our agent is trained with 8 A100 GPUs for 72 hours. We also compare with Cardsformer [9], the previous state-of-the-art self-play RL agent for Hearthstone. They train a model via supervised learning to predict game state transitions and use this prediction to train the policy model. We apply their method[3] and reproduce it on our larger scale of decks.

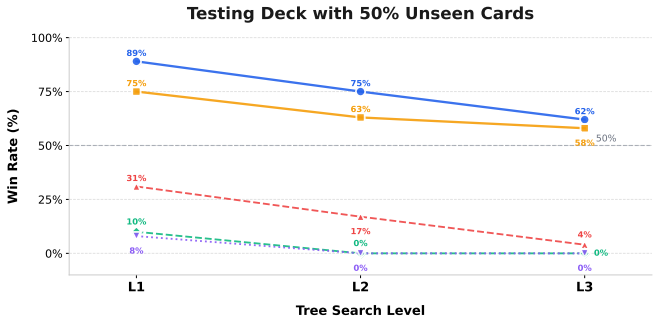
To compare these agents under identical conditions, we evaluate their strategic capability through mirror matches, where both players receive the same randomly selected deck from the 5,032 training decks, thereby eliminating deck matchup variance and directly measuring decision-making quality. Each agent pair is compared over 10,000 games (500 for GPT-5 mini due to inference cost), with standard deviations computed across 10 batches of 1,000 games.

As shown in Table III, our agent achieves win rates of 60.3%, 84.6%, and 93.8% against Cardsformer, GPT-5 mini, and the T5 agent, respectively, with consistently low standard deviations across all matchups. Notably, both GPT-5 mini and T5 agent fall behind both our agent and Cardsformer, indicating that self-play RL acquires game-specific strategic knowledge that prompting alone cannot capture.

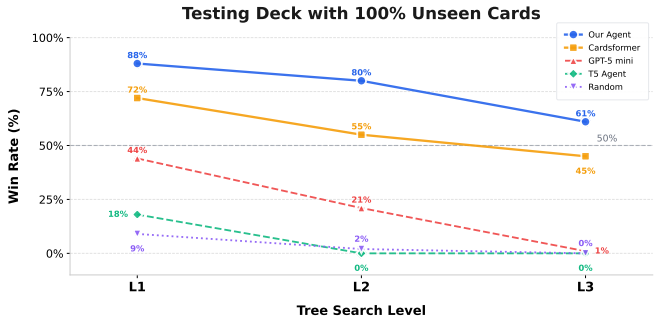
B. Generalization to Unseen Cards

While the mirror matches above evaluate decision-making on training decks, a key question is whether the learned strategies generalize to novel cards. We measure all agents against a tree search baseline, which simulates future moves by building a search tree over possible actions. Since tree search relies solely on forward simulation, its strategic performance is agnostic to whether the cards have been seen during training,

[3]https://github.com/WannianXia/Cardsformer



(a) The testing deck with 50% unseen cards



(b) The testing deck with 100% unseen cards

Fig. 3: Average win rates of different agents against tree search baselines on two testing decks with different amounts of novel cards.

making it a fair and consistent opponent for evaluating generalization. We define three difficulty levels: Level 1 (width 3, depth 10), Level 2 (width 5, depth 100), and Level 3 (width 10, depth 500), with Level 3 serving as the primary benchmark.

To test generalization, we design two testing decks with different amounts of novel cards our agent not trained on: one with 50% of the cards unseen and one with 100% of the cards unseen. For each deck, we run 100 games in total, with 50 games as first player and 50 as second player.

As shown in Fig. 3, our agent consistently outperforms all baselines across both settings and all difficulty levels. On the 50%-unseen deck, our agent achieves average win rates of 89%, 75%, and 62% against L1, L2, and L3 tree search agents, respectively. Cardsformer remains competitive but still falls behind our method by 4%–14%. On the fully-unseen deck, our agent maintains strong performance with win rates of 88%, 80%, and 61%, while Cardsformer shows a notable decline, particularly at L3 where it achieves only 45% compared to our 61%. The GPT-5 mini agent shows inconsistent performance, occasionally achieving reasonable results at L1 but dropping sharply at higher difficulty levels. Both the T5 agent and random policy fail to achieve meaningful success beyond L1. These results demonstrate that our approach enables robust strategic performance that transfers effectively to novel card configurations.

C. Error of Latent Dynamics

To evaluate the agent's ability to capture game transitions in latent space, we showcase two heatmaps of the MSE on sampled states in Fig. 4. In these figures, the x-axis represents 20 game states we randomly sampled from a game sequence. At each state, the agent performs an action, and the environment simulates the subsequent state. We then compared the hidden states $\hat{h}_{t+1}$ predicted by the latent dynamics module given the current observation and the action performed as $\hat{h}_{t+1} = w_\psi(f_\theta(o_t, a_t))$, with the hidden states derived from the representation module using the actual subsequent state as $\bar{h}_{t+1} = f_\theta(o_{t+1}, \text{Masked}(a_{t+1}))$. As each hidden state is a sequence of 32 observable entities (including hand cards, minions, heroes, and secrets) encoded by a transformer encoder, denoted as $h_t = \{h_t^i\}|_{i=1,2,\ldots,32}$, we computed the entity-wise MSE to quantify the results. Each entity within the same state is a data point along the y-axis. These results are shown in the left panel of Fig. 4. We also let the latent dynamics module infer the next latent representation with an incorrect random action, denoted as $\bar{h}'_{t+1} = w_\psi(f_\theta(o_t, a'_t))$, where a'_t is an incorrect action. We compare $\bar{h}'_{t+1}$ with the same labels $\bar{h}_{t+1}$ as in the left panel, so that we can demonstrate how the latent dynamics module is affected by the input action to get accurate predictions on future latent representations. These results are shown in the right panel of Fig. 4.

As shown in Fig. 4, when the latent dynamics module is given the right action, the majority of the entities across the sampled game states exhibit low error. The overall lighter shades in the left panel of Fig. 4 indicate a close alignment between the predicted latent representations and the ground-truth[4]. Conversely, there is a significant increase in MSE for many entities when the wrong actions are applied in the right panel of Fig. 4. These results demonstrate that the latent dynamics module can effectively predict game transitions in the latent space while maintaining sensitivity to action inputs, without overfitting to the output of the representation model.

D. Ablation Study

To assess the contribution of each component in our framework, we conduct an ablation study via head-to-head mirror matches (10,000 games per pair, variance computed over 10 intervals of 1,000 games). Each variant is trained for 24 hours (one-third of the full training budget) under the same hardware setup. We compare our full agent against three ablated variants:

- **w/o FT**: replaces the fine-tuned T5 encoder with the original pretrained T5-base;
- **w/o DS**: removes the deck strategy embedding from the policy module;
- **w/o LD**: removes the latent dynamics module and its auxiliary loss ($\beta = 0$).

[4]The so-called ground-truth are still bootstrapped latent representations from the representation module.

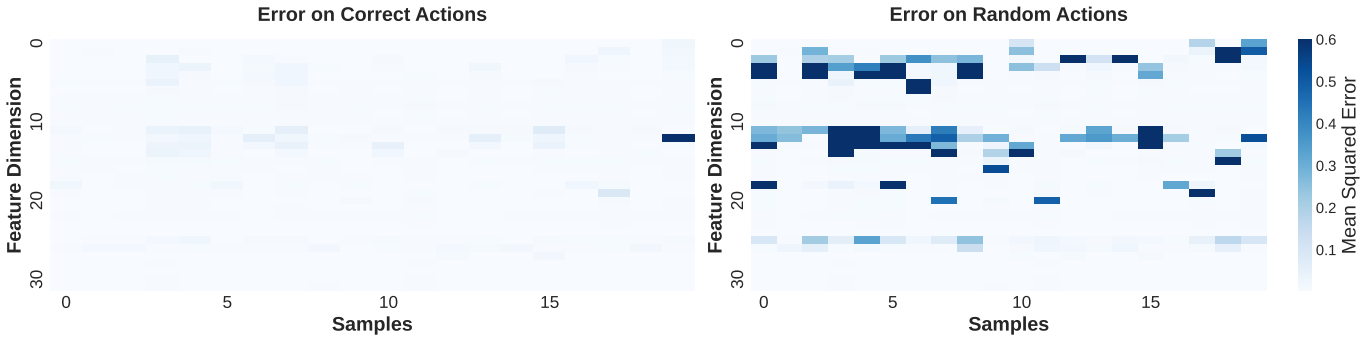


Fig. 4: The heatmaps of entity-wise transition loss of the latent dynamics. Each column in the heatmaps is the error of the latent dynamics module at a randomly sampled state.

TABLE IV: Ablation results on training decks. Each cell shows the win rate of the row agent against the column agent over 10,000 mirror match games.

	Our Agent	w/o FT	w/o DS	w/o LD
Our Agent	–	**73.9%**±4.1	**55.0%**±4.0	**89.5%**±1.2
w/o FT	26.1%±4.1	–	29.2%±3.4	65.3%±2.6
w/o DS	45.0%±4.0	70.8%±3.4	–	78.2%±2.1
w/o LD	10.5%±1.2	34.7%±2.6	21.8%±2.1	–

As shown in Table IV, the full model consistently outperforms all ablated variants. Removing the latent dynamics module (w/o LD) causes the most severe degradation: Our Agent achieves an 89.5% win rate against it, and it loses to every other variant. This confirms that the auxiliary latent dynamics loss is essential for efficient policy learning. Removing fine-tuning (w/o FT) leads to the second-largest drop, with Our Agent winning 73.9% against it, highlighting the importance of domain-specific fine-tuning of the language encoder. Removing the deck strategy embedding (w/o DS) has a comparatively smaller effect, yet Our Agent still holds a 55.0% win rate advantage, indicating that deck-level semantic information provides a meaningful signal for policy learning in Hearthstone.

E. Training Efficiency

While the ablation study above compares final performance, we further evaluate how quickly each variant learns. To evaluate this, we record each agent's win rate against a fixed random opponent throughout training. All variants, including Cardsformer, are trained under the same hardware setup (8×A100 GPUs) for 50 million frames. We evaluate every 1×10^6 training frames using 1,000 mirror matches against the random agent, and report the mean and ±1 standard deviation across 3 independent training seeds.

As shown in Fig. 5, our agent exhibits the fastest convergence among all methods, reaching 90% win rate at approximately 17×10^6 environment steps. The closest ablated variant (*w/o DS*) requires roughly 19×10^6 steps to reach the same level. Removing fine-tuning (*w/o FT*) further slows convergence to approximately 25×10^6 steps and Cardsformer shows a similar trajectory with (*w/o FT*). Most notably, the *w/o LD* variant fails to reach 90% win rate within the entire 50×10^6-step training horizon, demonstrating that the latent dynamics auxiliary loss is the critical component for efficient learning.

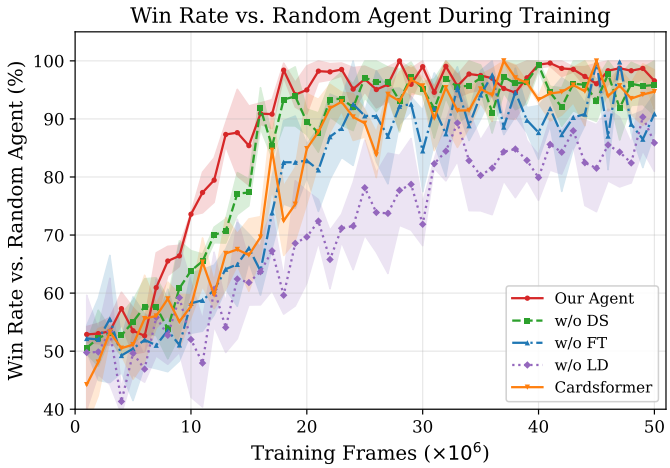


Fig. 5: Win rate against a random agent during training. Each data point represents an evaluation of 1,000 games at every 1×10^6 environment steps. Shaded regions denote ±1 standard deviation over 3 seeds.

IV. Related Work

Card games like Hearthstone have gained significant attention in recent AI research. In [20]–[22], researchers employed methods such as evolutionary algorithms and tree search to design Hearthstone AI. More recently, Xiao et al. [8] explored deep reinforcement learning to train agents that reached human-level performance. However, they overlooked the fact that Hearthstone cards are encoded with rich semantic information. They used one-hot encoding to represent the cards, thus ignoring the linguistic features within the game, resulting in lower training efficiency and an inability to generalize. Xia et al. [9] introduced the use of language models to represent card descriptions in Hearthstone, demonstrating generalization on unseen cards. Despite these advancements,

these aforementioned works were conducted on a limited number of cards, typically 100 to 300 training cards, thus narrowing the scope of their findings.

LLMs have also been investigated in language-embodied games. The FAIR team [23] developed Cicero, an AI agent that combines LLMs with planning and RL to achieve human-level performance in the strategy game Diplomacy. Xu et al. [24] and Xu et al. [25] performed tuning-free prompting and RL on LLMs to enhance agents' decision-making and strategic abilities in the communication game Werewolf. Park et al. [26] introduced generative agents powered by LLMs that simulate believable human behaviors, enabling interactive applications and emergent social behaviors in sandbox environments.

Our approach draws inspiration from recent model-based methods [27], particularly a series of works on the Dreamer framework [14]–[17], which implements agent interaction with a learned world model on hidden states to learn the policy. However, unlike Dreamer, our method prevents direct agent interaction with the world model. Instead, we introduce an auxiliary loss in the latent space similar to their training of the world model. We demonstrate in our experiments that auxiliary loss alone is highly efficient for the agent's learning process.

V. Conclusions

In this work, we propose a novel framework that integrates a fine-tuned LLM with RL to address the challenges of policy learning in the language-embodied textual card game Hearthstone. Our approach leverages a fine-tuned T5-base model to interpret and encode complex game interactions described in natural language. By introducing an auxiliary transition loss in the latent space, we further enhance the agent's ability to learn game policies across various scenarios efficiently. Our design demonstrates strong effectiveness in solving a complex textual card game like Hearthstone. We observe that our agent can outperform tree search baselines and other LLM-based agents with very limited training overhead, demonstrating the efficiency of our method. For future work, we plan to explore the use of generative LLMs as game agents, leveraging their reasoning and generation capabilities to further advance decision-making in complex game environments.

References

[1] K. F. Hubert, K. N. Awa, and D. L. Zabelina, "The current state of artificial intelligence generative language models is more creative than humans on divergent thinking tasks," *Scientific Reports*, vol. 14, no. 1, p. 3440, 2024.

[2] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen *et al.*, "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nature Machine Intelligence*, vol. 5, no. 3, pp. 220–235, 2023.

[3] N. Fei, Z. Lu, Y. Gao, G. Yang, Y. Huo, J. Wen, H. Lu, R. Song, X. Gao, T. Xiang *et al.*, "Towards artificial general intelligence via a multimodal foundation model," *Nature Communications*, vol. 13, no. 1, p. 3094, 2022.

[4] F. de Mesentier Silva, R. Canaan, S. Lee, M. C. Fontaine, J. Togelius, and A. K. Hoover, "Evolving the hearthstone meta," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.

[5] A. Summerville and M. Mateas, "Mystical tutor: A magic: The gathering design assistant via denoising sequence-to-sequence learning," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 12, no. 1, 2016, pp. 86–92.

[6] K. Chatterjee and R. Ibsen-Jensen, "The complexity of deciding legality of a single step of magic: The gathering," in *ECAI 2016*. IOS Press, 2016, pp. 1432–1439.

[7] M.-A. Côté, A. Kádár, X. Yuan, B. Kybartas, T. Barnes, E. Fine, J. Moore, R. Y. Tao, M. Hausknecht, L. E. Asri, M. Adada, W. Tay, and A. Trischler, "Textworld: A learning environment for text-based games," *CoRR*, vol. abs/1806.11532, 2018.

[8] C. Xiao, Y. Zhang, X. Huang, Q. Huang, J. Chen *et al.*, "Mastering strategy card game (hearthstone) with improved techniques," in *2023 IEEE Conference on Games (CoG)*. IEEE, 2023, pp. 1–8.

[9] W. Xia, Y. Yang, J. Ruan, D. Xing, and B. Xu, "Cardsformer: Grounding language to learn a generalizable policy in hearthstone," in *ECAI 2023*. IOS Press, 2023, pp. 2720–2727.

[10] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," in *Advances in Neural Information Processing Systems*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds., vol. 20. Curran Associates, Inc., 2007. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2007/file/08d98638c6fcd194a4b1e6992063e944-Paper.pdf

[11] G. Farina, C. Kroer, and T. Sandholm, "Regret circuits: Composability of regret minimizers," in *International conference on machine learning*. PMLR, 2019, pp. 1863–1872.

[12] N. Brown and T. Sandholm, "Superhuman ai for heads-up no-limit poker: Libratus beats top professionals," *Science*, vol. 359, no. 6374, pp. 418–424, 2018.

[13] ——, "Superhuman ai for multiplayer poker," *Science*, vol. 365, no. 6456, pp. 885–890, 2019.

[14] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=S1lOTC4tDS

[15] D. Hafner, T. P. Lillicrap, M. Norouzi, and J. Ba, "Mastering atari with discrete world models," in *International Conference on Learning Representations*, 2021.

[16] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse control tasks through world models," *Nature*, pp. 1–7, 2025.

[17] J. Lin, Y. Du, O. Watkins, D. Hafner, P. Abbeel, D. Klein, and A. Dragan, "Learning to model the world with language," 2023.

[18] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *International conference on machine learning*. PMLR, 2018, pp. 1407–1416.

[19] D. Zha, J. Xie, W. Ma, S. Zhang, X. Lian, X. Hu, and J. Liu, "Douzero: Mastering doudizhu with self-play deep reinforcement learning," in *international conference on machine learning*. PMLR, 2021, pp. 12 333–12 344.

[20] M. Świechowski, T. Tajmajer, and A. Janusz, "Improving hearthstone ai by combining mcts and supervised learning algorithms," in *IEEE Conference on Computational Intelligence and Games*, 2018, pp. 1–8.

[21] J. S. B. Choe and J.-K. Kim, "Enhancing monte carlo tree search for playing hearthstone," in *2019 IEEE Conference on Games*. IEEE, 2019, pp. 1–7.

[22] P. García-Sánchez, A. Tonda, A. J. Fernández-Leiva, and C. Cotta, "Optimizing hearthstone agents using an evolutionary algorithm," *Knowledge-Based Systems*, vol. 188, p. 105032, 2020.

[23] M. F. A. R. D. T. (FAIR)†, A. Bakhtin, N. Brown, E. Dinan, G. Farina, C. Flaherty, D. Fried, A. Goff, J. Gray, H. Hu *et al.*, "Human-level play in the game of diplomacy by combining language models with strategic reasoning," *Science*, vol. 378, no. 6624, pp. 1067–1074, 2022.

[24] Y. Xu, S. Wang, P. Li, F. Luo, X. Wang, W. Liu, and Y. Liu, "Exploring large language models for communication games: An empirical study on werewolf," *arXiv preprint arXiv:2309.04658*, 2023.

[25] Z. Xu, C. Yu, F. Fang, Y. Wang, and Y. Wu, "Language agents with reinforcement learning for strategic play in the werewolf game," *arXiv preprint arXiv:2310.18940*, 2023.

[26] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proceedings of the 36th annual acm symposium on user interface software and technology*, 2023, pp. 1–22.

[27] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.