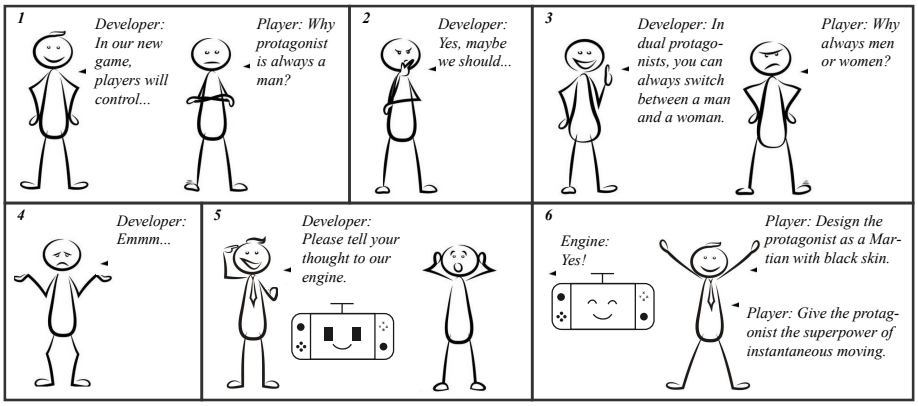

Figure 1: 1: Players were tired against the game's protagonist models. 2, 3: Developers thus created a new mode with dual protagonists. Players still didn't buy it, while they didn't know how to develop games. 4: There were irreconcilable divides between players and developers. 5, 6: Till the advent of the IDGE, it can read the players' mind and let them experience the games immediately.

Instruction-Driven Game Engines on Large Language Models

Hongqiu Wu[1], **Y. Wang**[*], **Xingyuan Liu**[1], **Hai Zhao**[1*], **Min Zhang**[2]
[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University
[2]Harbin Institute of Technology, Shenzhen
wuhongqiu@sjtu.edu.cn, yanwang.branden@gmail.com

Abstract

The *Instruction-Driven Game Engine (IDGE)* project aims to democratize game development by enabling a large language model (LLM) to follow free-form game rules and autonomously generate game-play processes. The IDGE allows users to create games by issuing simple natural language instructions, which significantly lowers the barrier for game development. We approach the learning process for IDGEs as a *Next State Prediction* task, wherein the model autoregressively predicts in-game states given player actions. It is a challenging task because the computation of in-game states must be precise; otherwise, slight errors could disrupt the game-play. To address this, we train the IDGE in a curriculum manner that progressively increases the model's exposure to complex scenarios.

Our initial progress lies in developing an IDGE for Poker, a universally cherished card game. The engine we've designed not only supports a wide range of poker variants but also allows for high customization of rules through natural language inputs. Furthermore, it also favors rapid prototyping of new games from minimal samples, proposing an innovative paradigm in game development that relies on

[*]Corresponding authors

minimal prompt and data engineering. This work lays the groundwork for future advancements in instruction-driven game creation, potentially transforming how games are designed and played.[1][2]

1 Introduction

Game developers dedicate creativity to offer immersive experiences to game players. Players immerse themselves in games and offer valuable feedback to developers. This makes a symbiotic relationship between creators and customers. However, as depicted in Figure 1, significant disconnections persist, due to diverse preferences of players across age, gender, and cultural backgrounds. Despite the fact that many today's games allow for customization of characters and appearances, it is an impossible task for developers to craft every aspect of the game to suit the need of every player. Our study seeks to reconcile such a divide.

Game engines, as the heart of game development, are conventionally driven by complex programming languages. This technical barrier often deters enthusiasts from realizing their game development dreams. In response, we propose a novel concept: *Instruction-Driven Game Engine (IDGE)*. This engine is designed to be instructable and scalable, enabling anyone to fashion a new game simply by providing natural language instructions. Distinct from recent advancements in video-based Game AI, such as CRADLE Tan et al. (2024) and SIMA DeepMind (2024), our focus in this paper is on the text-based prediction of game states, and leveraging Unity to render these text-described states to visually display.

IDGE is a neural engine, meaning it is built upon neural networks, specifically large language models (LLMs) (Brown et al., 2020; OpenAI, 2023; Touvron et al., 2023; Yang et al., 2023). It is designed to follow a **game script** - a detailed instruction that portrays the game settings, rules, elements, etc. - and drive the progression of game-play as interacting with players. IDGEs frame the operation of engines as a *Next State Prediction* task, autoregressively predicting the next in-game state based on the user-specified game script, previous in-game state, and current player action.

Training an IDGE faces the dual challenges of **stability** and **diversity**. The former seeks to provide a stable and precise game-play throughout lengthy contexts, while the latter seeks to follow diverse preferences within large player base. This necessitates that IDGE must adeptly navigate through a diverse set of game scripts while driving the games stably. Unfortunately, we empirically see a somewhat ironic twist: the model trained directly from existing game data seems to be neither stable nor diverse. Therefore, we employ a standard-to-diverse curriculum learning methodology to gradually introduce complexity into the training process. This strategy is designed to incrementally enhance the model's diversity while preserving its stability.

While it is still on journey from building an IDGE capable of producing AAA games, this paper provides our initial progress on **Poker**, a worldwide card game, e.g. *Texas hold'em*, *Badugi*. We train the IDGE using data sourced from a poker simulator. We show that the IDGE's understanding of nuanced semantics successfully fills voids left by the simulator program, e.g. suits, numbers, and game flow that never occurred in the training process. We further show its great potential to generalize to brand new game scripts, e.g. new card combinations and battle strategies, by few-shot learning and continue learning, a bold and promising idea for future game development process.

[1]Demo: `https://www.bilibili.com/video/BV1dA4m1w7xr/`; `https://youtu.be/jHTluHxJhqE`

[2]Repo: `https://github.com/gingasan/idge`

We summarize our paper below: • § 2 presents the concept of IDGE and formulates it as a learnable next state prediction task; • § 3 discusses the game-style data for the poker engine; • § 4 proposes the specific training techniques to enhance the training.

2 Game Engine

In this section, we introduce the dialogue-style LLM as the setup for *IDGEs*. Then, we discuss how we formulate the learning of IDGEs as a *Next State Prediction* problem.

2.1 From Instruction-Driven Dialogue to Instruction-Driven Game Engine

LLMs shape their knowledge by navigating massive amounts of human data. Most LLMs have been fine-tuned on dialogue-style corpora, where they are endowed with the ability to interact with users. The resultant models, such as ChatGPT (OpenAI, 2023), can follow a system instruction provided by users, and generate responses in line with the instructions during interaction.

Likewise, a game engine works through interaction, too. For an IDGE, the system instruction specifically refers to a game script that accurately describes the desired game. In game-play, the IDGE follows the predefined game script and interacts with players, concurrently processing player inputs, such as moves and targets, to dynamically generate the in-game states as responses.

In Figure 2, we demonstrate how a poker IDGE facilitates a variant of *Texas hold'em*: the user (or player) first inputs the game rules as the game script, with some specific rules described in natural language (described in the "Specific Rules" part), with a variation from standard *Texas hold'em* game. In game-play, following this game script, the IDGE computes and returns the game-play process state by state, with player actions, e.g. check, call, raise, till the game concludes. More technical details about how we infer these states will be introduced in § 4.

2.2 Next State Prediction

Causal language models learn the interplay of words and phrases through the autoregressive process of next token predicting (Vaswani et al., 2017; Brown et al., 2020). From a game-play perspective, the minimum component is no single token, but rather each in-game state. An in-game state is a single frame that contains all current game status, e.g. characters, items, and missions. Essentially, the task of any game engines is exactly to compute the next state according to prior ones. Therefore, we may formulate the learning of IDGEs as a *Next State Prediction (NSP)* problem.

Given a sequence of in-game states $\mathbf{s} = \{s_0, s_1, \cdots, s_T\}$, an IDGE with parameters θ seeks to maximize the likelihood: $\sum_{t=1}^{T} \log p_\theta(s_t|s_0, s_1, \cdots, s_{t-1}, x_t, z)$ where x_t refers to the player input at time t, and z refers to the game script which is global for the entire game. The engine seeks to predict the next state given the prior states following z.

An in-game state is typically much bigger than a single token, incurring overflow of input and weakness of long-range capture for language models (Beltagy et al., 2020; Xiao et al., 2023). A relaxed case occurs when it is assumed that each state s_t solely depends on its previous k states. Specifically when $k = 1$, the former equation can be reduced to:

$$\sum_{t=1}^{T} \log p_\theta(s_t|s_{t-1}, x_t, z). \tag{1}$$

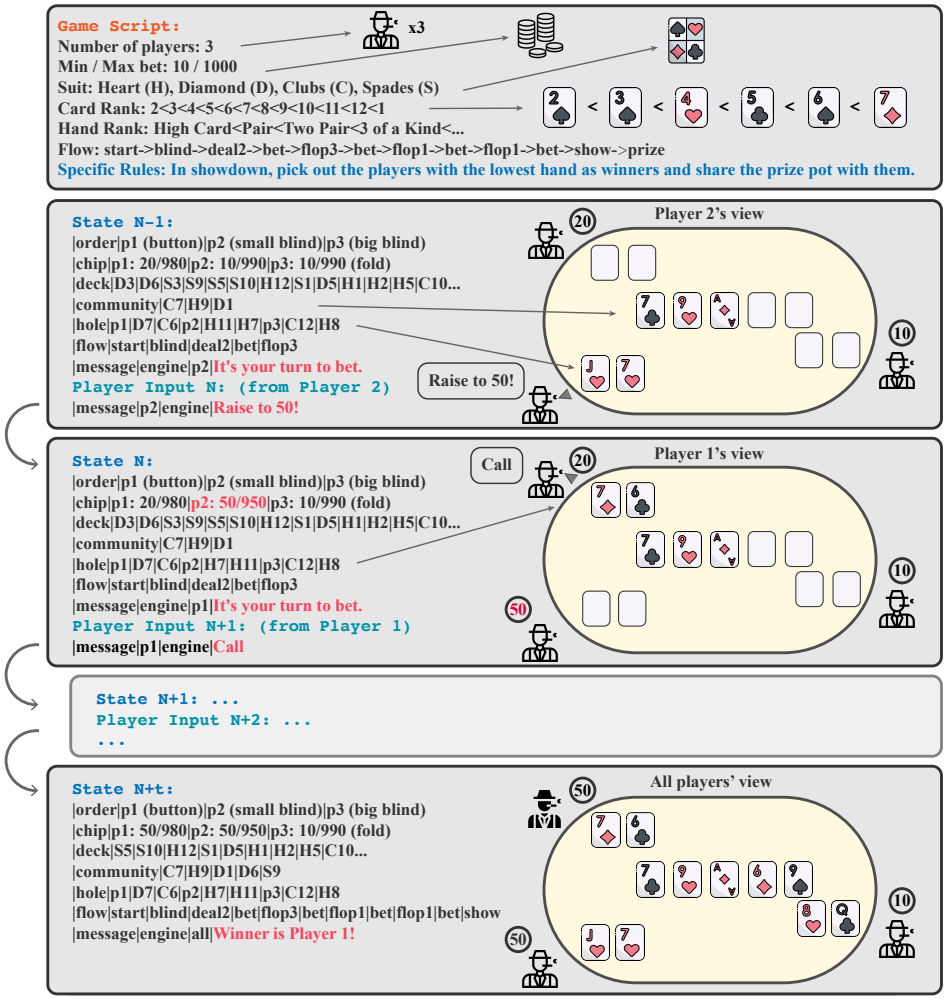

Figure 2: Game-style samples for next state prediction. The left side is the input text for the engine from a global view, including all parts that are visible to players as well as those that are not. The right side is the diagram of the game from different views.

While such an independence assumption would incur information loss, fortunately for a game engine, this loss can be avoided by the design of in-game states. We will discuss a concrete example in the following section.

3 Data for IDGE

In this paper, we generate a large number of game logs from a poker simulator. The simulator primarily supports ten representative types of poker games: *Texas hold'em*, *5-card draw*, *Omaha*, *Short-deck hold'em*, *2-to-7 triple draw*, *A-to-5 triple draw*, *2-to-7 single draw*, *Badugi*, *Badeucey*, and

Table 1: Statistics of training data.

# of samples	# of structured	# of natural	# of poker	avglen. (script)	avglen. (state)	vocab size	avg. players	avg. bet (min/max)	avg. states
100k	49.8k	50.2k	10	208.4	306.3	1120	4.6	6.0/1841	35.3

Badacey. Additionally, it allows for further configuration of several common elements for each poker game, including the number of players, the types of suits, the ranking of single-cards, the ranking of multi-card combinations, minimum and maximum bet limits, and the game flow, as shown in Figure 2. Detailed explanations of these functions can be found in Appendix F. By adjusting these common elements, one can derive virtually infinite variations beyond the aforementioned ten representative poker games. Importantly, in our dataset, each game corresponds to a unique configuration, which augments the model's ability to follow various game scripts.

Moreover, we realize that if the game logs are sampled completely in uniform, the occurrence of some rare states, e.g. card combinations, would be extremely low. The resultant engine trained on such data may fall short in low-frequency situations, even though the dataset is large. Therefore, we balance the data by up/down-sampling the game logs to ensure that all possible situations occur similarly. We show a concrete instance of our balancing process in Appendix D.

After obtaining game logs, we transform each log into a training sample as shown in Figure 2, for Next State Prediction (NSP). Each sample is made up of three parts: the game script z (structured or natural language), player input x_t, and in-game states s_t. If we were to draw an analogy with ChatGPT, these three parts respectively play the roles of the system, user, and assistant.

♠ **Game Script** We design a structured template for the game script to represent the customized configuration of each game, as depicted by the black text in the top part of Figure 2.

In addition to the structured script, another part of the game script includes the description of specific game rules in natural language. The structural script can fully describe the variants of only two out of the ten types of games: *Texas Hold'em* and *5-card draw*. For the remaining eight types of games, we utilize natural language to describe the aspects not covered by the structural script. For instance, the blue text within the top part of Figure 2 corresponds to the manually written specific rules of the game *2-to-7 triple draw*. Additionally, for each game, we use GPT3.5 to paraphrase its description, to enhance the IDGE's eventual generalizability to different game scripts. The prompt we use is in Appendix C.

♠ **In-Game State and Player Input** For the in-game state and player input, we design a standardized language to represent them precisely and in short. As shown in the left side of Figure 2, "|deck" is followed by the remaining cards in the pile, while "|message|a|b" is followed by the message sent from a to b. Specifically, player 2 chooses to raise the bet. Given both the previous states and player action as input, the engine outputs a new state, where the chips of player 2 are updated and player 1 is informed to bet since player 3 has folded.

To ensure the independence assumption that each state s_t solely depends on its previous state s_{t-1}, we design the in-game state to include all the information required for computing the next state. As a result, regardless of the amount of history information, the engine can always precisely represent the game status by updating the above elements.

Data Statistics Table 1 shows the eventual statistics of the training data. We obtain 100k samples from the simulator, where structured and natural language scripts are nearly equal. The average number of states of one simulated round is 35.3, i.e. the number of states for the engine to predict.

4 Curriculum Learning

Straightforwardly, we could utilize the standard data generated in § 3 to fine-tune a base model by maximizing Eq. 1 and obtain the IDGE. However, it may struggle with stability and diversity: neither can it accurately predict the next state nor comprehend the game script specified by users in natural language. Therefore, we devise a progressive curriculum learning process (Bengio et al., 2009), to incrementally enhance the IDGE's diversity while preserving stability.

WARMUP: Training with Core Set A game engine encompasses a complex collection of functionality. For example, a poker engine should deal, flop, and switch cards with deftness. The cold start problem is highlighted when all these sub-tasks are thrust into the model at once. To this end, we propose a pre-learning phase to warmup the engine. We define a *Core Set (CS)*, a collection of fundamental functions that form the backbone capabilities of the engine, e.g. dealing a certain number of cards, listing all possible combinations from the given cards. We derive an instruction tuning dataset from it, where each sample is a single instruction of a poker function. We heuristically craft 40 functions in the core set (see Appendix H).

The core set is akin to a basic library in most computer systems, while each function is an instruction described in natural language and the LLM learns these functions as a way of instruction following.

STANDARD: Training on Standard Game Scripts The next step is to train the model on the standard data introduced in § 3 by optimizing NSP. In this process, the task is to learn to become an engine by following the game scripts, combining pre-learned core functions organically.

DIVERSE: Training on Rephrased Game Scripts While our standard data already includes some game scripts with natural language description, the majority of game scripts are still structured in nature. Mastering structured description can be cumbersome and too strict for users. Rather, it is more natural for them to describe the desired game in natural language. Rather than exhaustively crafting new natural language data, we propose *Segment Rephrasing (SR)*, a technique that rephrases a portion of the structured description into natural language to encourage the model to follow diverse natural language game scripts.

As we introduced in § 3, the structured part of the game script contains seven elements. We randomly sample several of them and rephrase them into natural language. An example is shown in Table 5 in Appendix. To largely keep the semantics intact, there is only a very low probability that the entire script will be rephrased. The rephrasing process is done by GPT3.5 (in Appendix C). These rephrased game scripts will be more challenging to understand. The resultant model acquires the capability to follow sophisticated game scripts accurately (natural language or a mixture of structured and natural language). Readers may refer to Table 3 in § 5 for a concrete example.

We eventually summarize the training pipeline as follows: 1) train on a 10k core set $\mathcal{D}_E$; 2) train by optimizing NSP on the standard dataset $\mathcal{D}_M$ (100k); 3) train by optimizing NSP on 10k segment rephrased samples $\mathcal{D}_H$ and 10k standard samples.

The **warmup**, **standard**, and **diverse** process correspond to the easy, medium, and hard curriculum. It is a smooth transfer of the IDGE from standardization to fully instructable.

5 Experimental Results

We evaluate the IDGE on two scenarios: in-domain and out-of-domain games. The former is automatically generated by our simulator, which can be considered as a test set that has the same distribution as training. The latter resembles real-world situations, where proficient poker players

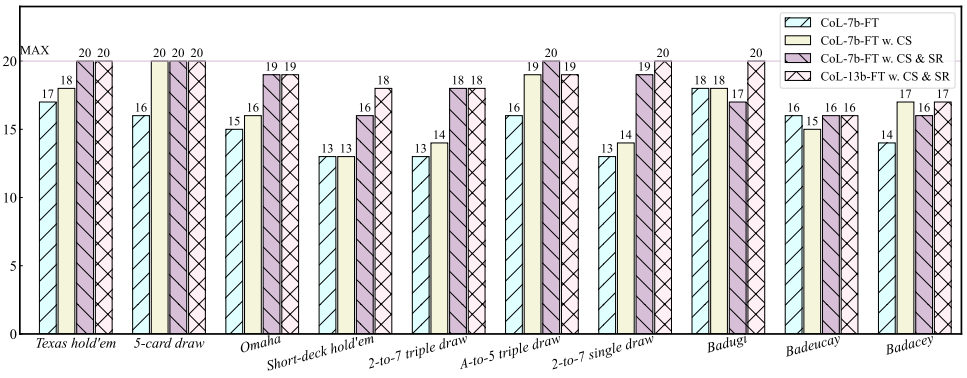

Figure 3: Round-level success rates on variants of 10 prototype poker games. In addition, our experiment turns out both GPT3.5 and GPT4 only achieve a zero success rate on all these games.

are directly enlisted as annotators to create new game scripts. Subsequently, games are played based on these new scripts to obtain test data. Ablation study is reported in Appendix E.

5.1 Training and Evaluation Setup

We train each model using LoRA (Hu et al., 2022) with $r = 8$, $\alpha = 32$, and the optimal learning rate in 1.5e-4 and 3e-4. We set the warmup steps to 30 and total batch size to 8 for 8 chips. For each curriculum, we train 5 epochs. To ensure the stability of outputs, we leverage greedy decoding.

• **In-domain evaluation**: The model has been exposed to a broad range of variants based on ten existing poker games during training. We sampled some unseen variants of these ten games from the poker simulator for evaluation. Then, we program some random players that randomly select an action as their input, to play games on IDGE. In such a scenario we can quickly evaluate the correctness of IDGE's predicted states using the poker simulator. There are 200 games in total in our in-domain evaluation set.

• **Out-of-domain evaluation**: The in-domain game scripts allow for the configuration of only seven key elements. To evaluate our engine performance in scenarios more closely aligned with the real world, we further recruit five proficient poker players as our engine testers. Each of them is asked to create a new game based on their personal preferences using natural language. They are free to tailor the game scripts, but these scripts should not be entirely unrelated to *Texas hold'em* or *5-card draw*. Subsequently, we invite them to play ten rounds for each new game and record all player inputs and in-game states throughout the game-play. This forms our out-of-domain evaluation set that comprises five distinct game scripts and 50 rounds of games.

5.2 In-Domain Games

Round-level Figure 3 depicts the round-level success rates of fine-tuned CodeLLaMA (Rozière et al., 2023) (CoL). The success rate is counted if the engine correctly handles all states in a round.

In our results, the most surprising aspect is the performance of GPT3.5 and GPT4. In 200 rounds of games, they are unable to successfully complete any single round. In contrast, fine-tuned models produce satisfactory results. Even the vanilla CoL-7b-FT model achieves an average success rate. Utilizing the core set (CS) to warmup further improves the performance, outperforming the vanilla

Table 2: State-level performances with different training methods. ⋆ refers to the model w. CS & SR.

	start	blind	deal	flop	switch	bet	show	prize	avg.
GPT3.5 - 5-shot	74.0	60.5	0	6.0	6.9	46.0	40.5	78.5	39.1
GPT4 - 5-shot	68.0	84.0	0	0	11.7	20.6	48.0	**92.0**	40.5
CodeLLaMA-7b - FT	100	100	90.0	100	98.9	98.3	80.5	99.0	95.8
w. *Core Set Warmup*	100	100	**100**	100	99.6	**99.7**	91.0	100	98.8
w. *Segment Rephrasing*	100	100	100	100	**100**	98.5	**92.0**	100	98.8
LLaMA2-7b - FT⋆	100	100	98.0	100	96.0	96.6	78.5	96.0	95.6
CodeLLaMA-13b - FT⋆	100	100	100	100	100	100	95.5	100	**99.4**

one in seven out of ten game variants. Furthermore, the model underwent full curriculum (w. CS & SR) showcases a remarkable improvement, e.g. +5 on *2-to-7 triple draw* and +6 on *2-to-7 single draw*. This suggests that challenging rephrasing samples encourage the model to learn a better alignment between structured and natural language game scripts, thereby better balancing stability and diversity. Lastly, increasing the model size also leads to performance gain.

State-level One might question why GPT3.5 and GPT4 completely fail in this task, significantly behind fine-tuned CodeLLaMA. To conduct a more in-depth analysis, we compute the state-level accuracy corresponding to each function in Table 2[3]. We find that, though GPT3.5/4 are strong in mathematical calculation (`prize`), they perform poorly in remaining functions. The underlying fact is that they struggle to manage cards accurately. For example, they are very likely to mess up the order of cards, hallucinating new cards or missing some of them. We conjecture that they were not exposed to highly sophisticated data and tasks as for the engine during their training. Accumulation of errors in all these functions leads non-fine-tuned models to zero success rates in round-level evaluation. It is important to note that the engine is required to be all-round at each function; otherwise, the overall engine performance will degenerate in a way of **Buckets effect**.

For fine-tuned models, they perform close to 100% in most functions, which guarantees the engine's stable round-level performance as in Figure 3. CS shows particularly beneficial for two specific functions, i.e. +10 on `deal`, +10.5 on `show`, which we notice as challenging aspects for a poker engine. We also find that code pre-training is beneficial to IDGEs, i.e. CodeLLaMA works better than LLaMA2. We conjecture that processing structured inputs as in-game states shares similarity with processing code, which is also structured (Guo et al., 2021; Wu et al., 2021).

5.3 Out-of-Domain Games

In Table 3, from script 1 to 5, customization of the script becomes more, and the gap from standard scripts also becomes larger. For example, the most challenging script 5 portrays a brand new game with defining two novel six-hand combinations, "Three Pair" and "Big House".

We apply few-shot in-context learning to adapt the engine to new games. The in-context samples are also crafted by invited players. Instead of listing all samples of a complete round, which leads to lengthy input, we allocate them by their functions. In other words, for each function, we solely place its corresponding samples in its game script (example in Appendix I).

We report the results in Table 4 based on CoL-13b with and without SR. Intuitively, models incorporated SR significantly outperform those without SR across all five game scripts, suggesting

[3] `https://huggingface.co/meta-llama/Llama-2-7b-chat-hf`;`https://huggingface.co/codellama/CodeLlama-7b-Instruct-hf`;`https://huggingface.co/codellama/CodeLlama-13b-Instruct-hf`

Table 3: Out-of-domain game scripts. We skip some settings for brevity, e.g. the number of players. Particularly for script 3, we did not include the "All-in" operation in the training data.

Script 1: ***reverse ranking + less suits***
In this game of Texas hold'em, there are 3 players and the minimum bet is 2, maximum bet is 1000. **The card numbers rank** from low to high as follows: 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2. **The suits are D, O, and G.**
Script 2: ***additional dealing phase***
The game proceeds in the following order: start, blinds, deal 2 cards, bet, reveal 3 cards (the flop), bet, reveal 1 card (the turn), **deal 1 cards (new deal)**, bet, show, and finally the prize is distributed.
Script 3: ***all-in***
Define a new player operation in the phase of bet "All-in". The player puts all his remaining chips into the pot, and is no longer able to make further bet during the game.
Script 4: ***3-card draw***
Introduce a new game, named "3-card draw". In this game, there are 3 suits, H, D, C. In addition, define **two new combinations with 3 cards in hand.** "Straight": there are three consecutive cards, e.g. C10, H11, D12. "Flush": there are three cards within the same suit, e.g. H1, H10, H6. All combinations rank as: High Card<Pair<Three of a Kind<Straight<Flush<Straight Flush.
Script 5: ***6-card draw***
Introduce a new game, named "6-card draw". In this game, there are 5 suits, H, D, C, S, and R. In addition, define **two new combinations with 6 cards in hand.** "**Three Pair**": there are three pairs of distinct numbers, e.g. R8, H8, C10, H10, H12, D12. "**Big House**": there are two pairs of three of one kind, e.g. R8, H8, C8, D12, H12, D12. All combinations rank as: High Card<Pair<Three of a Kind<Straight<Flush<**Full House**<**Three Pair**<**Big House**<Straight Flush

Table 4: Success rates for 10 rounds (%) on out-of-domain games. CT refers to continue-training.

	w/o. *Segment Rephrasing*				w. *Segment Rephrasing*			
Script	**0-shot**	**3-shot**	**10-shot**	**CT**	**0-shot**	**3-shot**	**10-shot**	**CT**
1: *ranking + suits*	80	100	-	-	**100**	100	-	-
2: *additional dealing*	0	70	-	-	**80**	100	-	-
3: *all-in*	-	0	10	-	-	**70**	100	-
4: *3-card draw*	-	0	0	-	-	50	**80**	**100** (8-shot)
5: *6-card draw*	-	0	0	0	-	0	10	**100** (23-shot)

that SR is indispensable for the engine to understand pure natural language inputs. Script 3 and 4 present two challenging cases, as they define new operations or combinations. We observe that even models enhanced with SR struggle to accurately compute each state solely based on the scripts. Fortunately, when we provide it with some samples, the engine operating in a few-shot learning manner achieves quite satisfactory results. This is exactly a novel game development process introduced by IDGEs: we may shift the burden from writing new programs to crafting a detailed game script and a small number of samples, a process often referred as "prompt engineering".

Script 5 presents the most challenging case, pushing the engine's upper limit. Due to its complexity, the model can no longer accurately predict solely through in-context learning. Therefore, we explore

a new solution: **continue-training**. We ask players to manually label incorrectly predicted states while playing script 5 using the IDGE, and correct them before continuing the game. In this manner, we construct a number of pairs of "good states" and "bad states". We thus employ DPO (Rafailov et al., 2023) to optimize the engine. We find that it quickly learns to rectify its shortcomings guided by user feedback, eventually achieving satisfactory outcomes (achieving 100% with 8/23-shot for script 4/5). This provides a new perspective for application of IDGEs in real-world scenarios: users can customize their individual evolving IDGE by own feedback, continually refining it till satisfied.

6 Conclusion

This paper introduces the Instruction-Driven Game Engine, offering game enthusiasts a brand new game development and playing experience. We formulate the learning of IDGEs as Next State Prediction and leverage a curriculum learning approach to enhance stability and diversity. Experiments demonstrate the proposed engine can accurately complete the majority of user-defined games. For challenging cases, the engine can evolve with user feedback in an RLHF manner.

This paper presents our initial progress in Poker games. Such a paradigm theoretically applies to all types of games. However, our progress is constrained by several bottlenecks:

Inference Latency We have demonstrated that IDGEs go well with turn-based strategy (TBS) games. For real-time strategy (RTS) games, players may make more than one action per second. The inference latency of LLMs cannot meet the real-time requirements of such games.

Context Window Size Generally, as games become more complex, the length of in-game states increases, posing a challenge to satisfy our independent assumption. This may significantly challenge both the comprehension ability of LLMs and the cache of KV states.

Accessibility The kernel data of most commercial games is not publicly available, which is why we developed a poker simulator to generate the training data for this paper.

We are delighted to observe that there have been continuous advancements in inference frameworks such as vLLM (Kwon et al., 2023), as well as efficient long-text generation methods like StreamingLLM (Xiao et al., 2023) and Temp-LoRA (Wang et al., 2024). We believe that the ongoing development of LLM technologies will ultimately address the limitations of latency and the context window. Regarding the issue of accessibility, we look forward to more companies providing open interfaces as SC2LE (Vinyals et al., 2017), HOK Arena (Wei et al., 2022) to offer kernel game data.

References

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020. URL https://arxiv.org/abs/2004.05150.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman (eds.), *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, pp. 41–48. ACM, 2009. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Commun. ACM*, 60(11):81–88, 2017. doi: 10.1145/3131284. URL https://doi.org/10.1145/3131284.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Google DeepMind. Scaling instructable agents across many simulated worlds. 2024. URL https://storage.googleapis.com/deepmind-media/DeepMind.com/Blog/sima-generalist-ai-agent-for-3d-virtual-environments/Scaling%20Instructable%20Agents%20Across%20Many%20Simulated%20Worlds.pdf.

Mirjam Palosaari Eladhari. Re-tellings: The fourth layer of narrative as an instrument for critique. In Rebecca Rouse, Hartmut Koenitz, and Mads Haahr (eds.), *Interactive Storytelling - 11th International Conference on Interactive Digital Storytelling, ICIDS 2018, Dublin, Ireland, December 5-8, 2018, Proceedings*, volume 11318 of *Lecture Notes in Computer Science*, pp. 65–78. Springer, 2018. doi: 10.1007/978-3-030-04028-4_5. URL https://doi.org/10.1007/978-3-030-04028-4_5.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/74a67268c5cc5910f64938cac4526a90-Abstract-Datasets_and_Benchmarks.html.

Roberto Gallotta, Graham Todd, Marvin Zammit, Sam Earle, Antonios Liapis, Julian Togelius, and Georgios N Yannakakis. Large language models and games: A survey and roadmap. *arXiv preprint arXiv:2402.18659*, 2024.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. Graphcodebert: Pre-training code representations with data flow. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=jLoC4ez43PZ.

Akshat Gupta. Are chatgpt and GPT-4 good poker players? - A pre-flop analysis. *CoRR*, abs/2308.12466, 2023. doi: 10.48550/ARXIV.2308.12466. URL https://doi.org/10.48550/arXiv.2308.12466.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Neel Jain, Ping-yeh Chiang, Yuxin Wen, John Kirchenbauer, Hong-Min Chu, Gowthami Somepalli, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Neftune: Noisy embeddings improve instruction finetuning.

CoRR, abs/2310.05914, 2023. doi: 10.48550/ARXIV.2310.05914. URL https://doi.org/10.48550/arXiv.2310.05914.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *CoRR*, abs/2310.06825, 2023. doi: 10.48550/ARXIV.2310.06825. URL https://doi.org/10.48550/arXiv.2310.06825.

Juho Kim. Pokerkit: A comprehensive python library for fine-grained multi-variant poker game simulations. *CoRR*, abs/2308.07327, 2023. doi: 10.48550/ARXIV.2308.07327. URL https://doi.org/10.48550/arXiv.2308.07327.

Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/569ff987c643b4bedf504efda8f786c2-Abstract.html.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023.

Ryan Lowe, Abhinav Gupta, Jakob N. Foerster, Douwe Kiela, and Joelle Pineau. On the interaction between supervision and self-play in emergent communication. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rJxGLlBtwH.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Matej Moravcík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael H. Bowling. Deepstack: Expert-level artificial intelligence in no-limit poker. *CoRR*, abs/1701.01724, 2017. URL http://arxiv.org/abs/1701.01724.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/arXiv.2303.08774. URL https://doi.org/10.48550/arXiv.2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu

Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. Tool learning with foundation models. *CoRR*, abs/2304.08354, 2023. doi: 10.48550/ARXIV.2304.08354. URL https://doi.org/10.48550/arXiv.2304.08354.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2023.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Noah Ranella and Markus Eger. Towards automated video game commentary using generative AI. In Abdelrahman Madkour, Jasmine Otto, Lucas N. Ferreira, and Shi Johnson-Bey (eds.), *Proceedings of the Experimental Artificial Intelligence in Games Workshop co-located with the 19th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2023), Salt Lake City, Utah, USA, October 8, 2023*, volume 3626 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL https://ceur-ws.org/Vol-3626/paper7.pdf.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950, 2023. doi: 10.48550/ARXIV.2308.12950. URL https://doi.org/10.48550/arXiv.2308.12950.

Murray Shanahan, Kyle McDonell, and Laria Reynolds. Role play with large language models. *Nat.*, 623(7987):493–498, 2023. doi: 10.1038/S41586-023-06647-8. URL https://doi.org/10.1038/s41586-023-06647-8.

Weihao Tan, Ziluo Ding, Wentao Zhang, Boyu Li, Bohan Zhou, Junpeng Yue, Haochong Xia, Jiechuan Jiang, Longtao Zheng, Xinrun Xu, Yifei Bi, Pengjie Gu, Xinrun Wang, Börje F. Karlsson, Bo An, and Zongqing Lu. Towards general computer control: A multimodal agent for red dead redemption II as a case study. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024. URL https://openreview.net/forum?id=pmcFzuUxsP.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023. doi: 10.48550/arXiv.2307.09288. URL https://doi.org/10.48550/arXiv.2307.09288.

Muhtar Çagkan Uludagli and Kaya Oguz. Non-player character decision-making in computer games. *Artif. Intell. Rev.*, 56(12):14159–14191, 2023. doi: 10.1007/S10462-023-10491-7. URL https://doi.org/10.1007/s10462-023-10491-7.

Nidhi Vakil and Hadi Amiri. Complexity-guided curriculum learning for text graphs. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pp. 2610–2626. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.findings-emnlp.172.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft ii: A new challenge for reinforcement learning, 2017.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019. doi: 10.1038/S41586-019-1724-Z. URL https://doi.org/10.1038/s41586-019-1724-z.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *CoRR*, abs/2305.16291, 2023. doi: 10.48550/ARXIV.2305.16291. URL https://doi.org/10.48550/arXiv.2305.16291.

Y. Wang, D. Ma, and D. Cai. With greater text comes greater necessity: Inference-time training helps long text generation, 2024.

Hua Wei, Jingxiao Chen, Xiyang Ji, Hongyang Qin, Minwen Deng, Siqin Li, Liang Wang, Weinan Zhang, Yong Yu, Lin Liu, Lanxiao Huang, Deheng Ye, Qiang Fu, and Wei Yang. Honor of kings arena: an environment for generalization in competitive reinforcement learning, 2022.

Hongqiu Wu, Hai Zhao, and Min Zhang. Code summarization with structure-induced transformer. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pp. 1078–1090. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.FINDINGS-ACL.93. URL https://doi.org/10.18653/v1/2021.findings-acl.93.

Hongqiu Wu, Linfeng Liu, Hai Zhao, and Min Zhang. Empower nested boolean logic via self-supervised curriculum learning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 13731–13742. Association for Computational Linguistics, 2023a. URL https://aclanthology.org/2023.emnlp-main.847.

Hongqiu Wu, Yongxiang Liu, Hanwen Shi, Hai Zhao, and Min Zhang. Toward adversarial training on contextualized language representation. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023b. URL https://openreview.net/pdf?id=xZD10GhCvM.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *CoRR*, abs/2309.17453, 2023. doi: 10.48550/ARXIV.2309.17453. URL https://doi.org/10.48550/arXiv.2309.17453.

Yuzhuang Xu, Shuo Wang, Peng Li, Fuwen Luo, Xiaolong Wang, Weidong Liu, and Yang Liu. Exploring large language models for communication games: An empirical study on werewolf. *CoRR*, abs/2309.04658, 2023. doi: 10.48550/ARXIV.2309.04658. URL https://doi.org/10.48550/arXiv.2309.04658.

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, Fan Yang, Fei Deng, Feng Wang, Feng Liu, Guangwei Ai, Guosheng Dong, Haizhou Zhao, Hang Xu, Haoze Sun, Hongda Zhang, Hui Liu, Jiaming Ji, Jian Xie, Juntao Dai, Kun Fang, Lei Su, Liang Song, Lifeng Liu, Liyun Ru, Luyao Ma, Mang Wang, Mickel Liu, MingAn Lin, Nuolan Nie, Peidong Guo, Ruiyang Sun, Tao Zhang, Tianpeng Li, Tianyu Li, Wei Cheng, Weipeng Chen, Xiangrong Zeng, Xiaochuan Wang, Xiaoxi Chen, Xin Men, Xin Yu, Xuehai Pan, Yanjun Shen, Yiding Wang, Yiyu Li, Youxin Jiang, Yuchen Gao, Yupeng Zhang, Zenan Zhou, and Zhiying Wu. Baichuan 2: Open large-scale language models. *CoRR*, abs/2309.10305, 2023. doi: 10.48550/ARXIV.2309.10305. URL https://doi.org/10.48550/arXiv.2309.10305.

Enmin Zhao, Renye Yan, Jinqiu Li, Kai Li, and Junliang Xing. Alphaholdem: High-performance artificial intelligence for heads-up no-limit poker via end-to-end reinforcement learning. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pp. 4689–4697. AAAI Press, 2022. doi: 10.1609/AAAI.V36I4.20394. URL https://doi.org/10.1609/aaai.v36i4.20394.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. Freelb: Enhanced adversarial training for natural language understanding. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BygzbyHFvB.

Table 5: An example of segment rephrasing.

Game Script for *Texas hold'em*
Number of players: 3 (In this game of Texas hold'em, there are three players.) Card rank: 2<3<4<5<6<7<8<9<10<11<12<1 Min / Max bet: 10 / 5000 Flow: start->blind->deal2->bet->flop3->bet->flop1->... (The game begins with placing the blinds, followed by dealing 2 cards to each player, placing the bet for each...)

A Segment Rephrasing

B Related Work

A game engine is a fundamental software designed for the creation of games, providing developers with necessary tools. Famous game engines include Unreal, Unity, CryENGINE, etc. We spotlight two crucial properties of a game engine. The first is functionality, i.e. providing a wide variety of basic tools to facilitate the development process. The next is secondary development, i.e. rich and flexible interfaces to allow developers to customize new games. In this work, we introduce a new concept, instruction-driven game engine (IDGE), a neural game engine learned on basis of large language models (OpenAI, 2023; Touvron et al., 2023; Jiang et al., 2023; Yang et al., 2023; Qin et al., 2023). As opposed to a typical game engine, an IDGE acquires its functionality power by instruction tuning on the core set (Brown et al., 2020; Raffel et al., 2020; Ouyang et al., 2022) and allows for low-barrier game development by natural language description.

AI for games is an exciting area in AI research. A great amount of recent work studies learning for agents, e.g. as game players for Atari (Mnih et al., 2013), Minecraft (Fan et al., 2022; Wang et al., 2023), StarCraft, (Vinyals et al., 2019), NetHack (Küttler et al., 2020; Lowe et al., 2020), Werewolf (Xu et al., 2023); as non-play characters (NPCs) (Shanahan et al., 2023; Uludagli & Oguz, 2023); player assistants (Gallotta et al., 2024); game commentators (Eladhari, 2018; Ranella & Eger, 2023). Most of these work aims to train an AI player based on game logs or images. However, our work diverges from all of them in that we focus on the LLM as a game engine, attempting to build a game engine that is defined by instructions (game scripts) and data (in-game states). In addition, an agent focuses on the way AI behaves, while an engine focuses on the way AI would react in the face of any possible behaviors from any human being or agent. More recent work comes up with learning for a foundation agent, a single agent with generalizable skills to behave in various environments, e.g. SIMA (DeepMind, 2024), an instruction-driven agent proficient in multiple simulated environments; CRADLE (Tan et al., 2024), a powerful agent capable of playing complex AAA games like Red Dead Redemption 2 by controlling the keyboard and mouse. However, our work targets the IDGE for a specific group of games, Poker, as an initial step for building a foundation IDGE. Poker is a widely studied information game of immense popularity (Bowling et al., 2017; Moravcík et al., 2017; Gupta, 2023; Kim, 2023; Zhao et al., 2022).

In this paper, the entire training cycle for IDGE is a way of curriculum learning (Bengio et al., 2009). Recent studies show the promising role of curriculum learning in empowering the language models to tackle more challenging tasks (Vakil & Amiri, 2023; Wu et al., 2023a). The proposed segment rephrasing technique is related to perturbation training (Zhu et al., 2020; Wu et al., 2023b; Jain et al., 2023), which smooths the boundary of structured and natural language in the semantic space.

C Prompt used for GPT3.5

Table 6: Prompts for rephrasing structural and natural language.

Prompt for rephrasing structural language
You are a skilled English writer.
I will give you a paragraph that describes the rules of a Texas hold'em game, split by triple backticks.

```
{sent}
```

1. Use natural language to describe the rule.
2. It is important that do not change the cards.
3. You must reflect the ranking of the cards and do not change their order. This is also important.
4. Do not include additional information that is not reflected in the sentence.
5. Return the new sentence split by triple backticks.
6. Do not use "ace" in your response.

Input example 1
Flow: start->shuffle->blind->deal5->bet->switch->bet->switch->bet->show->prize
Output example 1
Process: The game starts with shuffling the cards, followed by assigning blinds, dealing five initial cards, placing bets, switching cards, placing more bets, switching again, placing bets once more, revealing the cards, and finally awarding the prize.
Input example 2
Card Rank: 7<6<2<14<9<11<4<17<16<1<8<3
Output example 2
The card numbers, from lowest to highest value, are 7, 6, 2, 14, 9, 11, 4, 17, 16, 1, 8, 3, and 10.

Prompt for rephrasing natural language
Paraphrase the given paragraph.

```
{sent}
```

1. It is important to keep your response as diverse as the input.
2. You should change the structure of the paragraph in your response.

Input example 1
In showdown, pick out the players with the lowest combination of cards as the winners.
Output example 1
In showdown, winners are determined as those players with the lowest aggregates of hand.
Input example 2
In showdown, define a new ranking strategy called `Badugi `. In given cards, pick out the cards of distinct suits and no pair. If there are more than one cards of the same suit or same value, choose the smaller-ranking one. Hence, the greatest Badugi refers to the the most number of distinct cards and the smallest values.
Output example 2
In the heat of showdown, draft a ranking outline known as `Badugi `. Choose cards that differ in suit and lack matching values. When selecting between cards of identical suits or values, opt for those with a lower rank. Hence, the epitome of a Badugi hand holds the most varied suits at the smallest card values.

D Data Efficiency

While the simulator program can generate massive data in a short period, we stress the importance of data efficiency. We find that a simple manipulation on the real data distribution effectively allows for a better training outcome with a smaller amount of training data. We use the combination of cards as an instance. Typically in a poker game, the chance of a straight (e.g. 5, 6, 7, 8, 9) in hand is much lower than a pair (e.g. 6, 6). As depicted in Figure 4 (left), recorded from 1,000 poker games, we find that the occurrences of "pair" and "high card" far outstrip the others. The model trained on such unbalanced data may fall short in low-frequency situations, even though the dataset is large. We thus balance the data sampling process so that each combination of cards occurs similarly in the game-play data, as in Figure 4 (right).

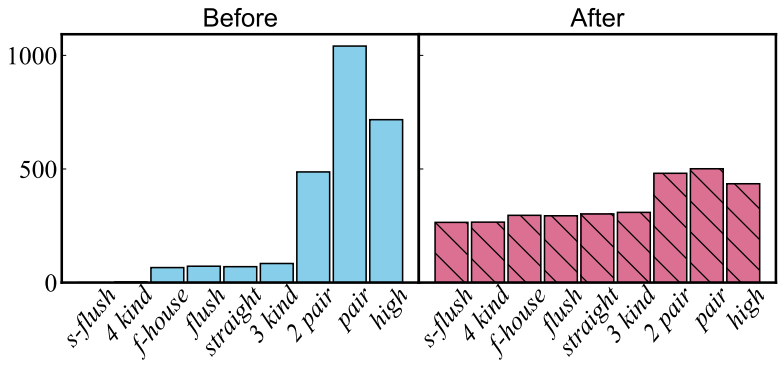

Figure 4: Balance the data distribution to boost data efficiency when sampling training data.

E Ablation Study

Table 7: Ablation study based on CoL-7b.

	`start`	`deal`	`show`	`prize`	*Script 1*	*Script 2*
Warmup + Standard w/o. data efficiency	99.5	84.5	72.5	98.5	60.0	0
Warmup + Standard	100	98.0	89.0	98.5	80.0	0
Warmup + Standard + Diverse w/o. down-sample	100	89.0	86.5	100	90.0	**90.0**
Warmup + Standard + Diverse	100	**100**	**92.0**	100	**100**	80.0

F Individual Function

Table 8: Reference for each individual function.

Flow	**Meaning**

Continued on next page

Table 8: Reference for each individual function. (Continued)

`start`	Order the cards and players, including the button, and deliver their chips if needed.
`blind`	Place the blind bet, e.g. for the big blind and small blind.
`dealx`	Deal x cards to each player.
`bet`	Ask each player to place the bet into the prize pool one by one.
`flopx`	Skip one card before turning over x cards from the deck.
`switch`	The player chooses to exchange some cards in his or her hand.
`show`	Show all hands of players and determine the winners.
`prize`	Split the prize pool to the winners.

G Game Script

Table 9: Game scripts of different poker games. We omit the number of players and bet limits.

Texas hold'em
Suit: Hearts (H), Diamonds (D), Clubs (C), Spades (S) **Card Rank:** 2<3<4<5<6<7<8<9<10<11<12<13<1 **Hand Rank:**High Card<Pair<Two Pair<Three of a Kind<Straight<Flush<Four of a Kind<Straight Flush **Flow:** start->blind->deal2->bet->flop3->bet->flop1->bet->show->prize
5-card draw
Suit: Hearts (H), Diamonds (D), Clubs (C), Spades (S) **Card Rank:** 2<3<4<5<6<7<8<9<10<11<12<13<1 **Hand Rank:**High Card<Pair<Two Pair<Three of a Kind<Straight<Flush<Four of a Kind<Straight Flush **Flow:**start->blind->deal5->bet->switch->bet->show->prize
Short-deck hold'em
Suit: Hearts (H), Diamonds (D), Clubs (C), Spades (S) **Card Rank:** 6<7<8<9<10<11<12<13<1 **Hand Rank:** High Card<Pair<Two Pair<Three of a Kind<Flush<Straight<Four of a Kind<Straight Flush **Flow:** start->shuffle->deal2->bet->flop3->bet->flop1->bet->flop1->bet->show->prize **Specific Rules:** Define a new combination "Small Straight". It allows the highest-ranking card to be used as the lowest-ranking when forming the straight. A small straight is smaller than a standard
Omaha

Continued on next page

Table 9: Game scripts of different poker games. We omit the number of players and bet limits. (Continued)

Suit: Hearts (H), Diamonds (D), Clubs (C), Spades (S)
Card Rank: 2<3<4<5<6<7<8<9<10<11<12<13<1
Hand Rank: High Card<Pair<Two Pair<Three of a Kind<Flush<Straight<Four of a Kind<Straight Flush
Flow: start->blind->deal4->bet->flop3->bet->flop1->bet->show->prize
Specific Rules:
In showdown, only a combination of 2 hole cards and 3 community cards can be used to form the optimal cards.

2-to-7 triple draw

Suit: Hearts (H), Diamonds (D), Clubs (C), Spades (S)
Card Rank: 2<3<4<5<6<7<8<9<10<11<12<13<1
Hand Rank: High Card<Pair<Two Pair<Three of a Kind<Straight<Flush<Four of a Kind<Straight Flush
Flow: start->blind->deal5->bet->switch->bet->switch->bet->switch->bet->show->prize
Specific Rules:
In showdown, pick out the players with the lowest combination of cards as the winners.

A-to-5 triple draw

Suit: Hearts (H), Diamonds (D), Clubs (C), Spades (S)
Card Rank: 1<2<3<4<5<6<7<8<9<10<11<12<13
Hand Rank: High Card<Pair<Two Pair<Three of a Kind<Full House<Four of a Kind
Flow: start->blind->deal5->bet->switch->bet->switch->bet->switch->bet->show->prize
Specific Rules:
In showdown, pick out the players with the lowest combination of cards as the winners.

Badugi

Suit: Hearts (H), Diamonds (D), Clubs (C), Spades (S)
Card Rank: 1<2<3<4<5<6<7<8<9<10<11<12<13
Flow: start->blind->deal4->bet->switch->bet->switch->bet->switch->bet->show->prize
Specific Rules:
In showdown, define a new ranking strategy called "Badugi". In given cards, pick out the cards of distinct suits and no pair. If there are more than one cards of the same suit or same value, choose the smaller-ranking one. Hence, the greatest Badugi refers to the the most number of distinct cards and the smallest values.

Badeucey

Continued on next page

Table 9: Game scripts of different poker games. We omit the number of players and bet limits. (Continued)

Suit: Hearts (H), Diamonds (D), Clubs (C), Spades (S) **Card Rank:** 2<3<4<5<6<7<8<9<10<11<12<13<1 **Hand Rank:** High Card<Pair<Two Pair<Three of a Kind<Full House<Four of a Kind **Flow:** start->blind->dea5->bet->switch->bet->switch->bet->switch->bet->show->prize **Specific Rules:** 1. In showdown, define a new ranking strategy called "Badugi". In given cards, pick out the cards of distinct suits and no pair. If there are more than one cards of the same suit or same value, choose the smaller-ranking one. Hence, the greatest Badugi refers to the the most number of distinct cards and the smallest values. 2. Pick out the players with the lowest combinations of cards and greatest Badugi as the winners, and split the prize pool equally, and one portion for the winners of the lowest combination, and the other portion for the winners of the greatest Badugi.
Badacey
Suit: Hearts (H), Diamonds (D), Clubs (C), Spades (S) **Card Rank:** 2<3<4<5<6<7<8<9<10<11<12<13<1 **Hand Rank:** High Card<Pair<Two Pair<Three of a Kind<Full House<Four of a Kind **Flow:** start->blind->dea5->bet->switch->bet->switch->bet->switch->bet->show->prize **Specific Rules:** 1. In showdown, define a new ranking strategy called "Badugi". In given cards, pick out the cards of distinct suits and no pair. If there are more than one cards of the same suit or same value, choose the smaller-ranking one. Hence, the greatest Badugi refers to the the most number of distinct cards and the smallest values. 2. Pick out the players with the lowest combinations of cards and greatest Badugi as the winners, and split the prize pool equally, and one portion for the winners of the lowest combination, and the other portion for the winners of the greatest Badugi.
2-to-7 single draw
Suit: Hearts (H), Diamonds (D), Clubs (C), Spades (S) **Card Rank:** 2<5<6<7<8<9<10<11<12<13<1 **Hand Rank:** High Card<Pair<Two Pair<Three of a Kind<Straight<Flush<Four of a Kind<Straight Flush **Flow:** start->blind->deal5->bet->switch->bet->show->prize **Specific Rules:** In showdown, pick out the players with the lowest combination of cards as the winners.

H Core Set

Table 10: Composition of the core set.

Function	Instruction
`shuffle`	Generate a deck of all cards and shuffle it following the settings.

Continued on next page

Table 10: Composition of the core set. (Continued)

`blind`	Set the blind players who are forced to place the bet. The small blind is the one to the left of the button player and the bet is half the minimum bet. The big blind is the one to the left of the small blind and the bet is the minimum bet.
`dealx`	Deal x cards to each of the players by order from the top of the deck.
`flopx`	Discard one card, and then flop x cards from the top of the deck as the community cards.
`switch`	Discard the specified cards, and then draw the same number of cards from the deck. This is specified by the user: 'pa: Switch x'.
`show`	Show all the hole cards of players and pick out one or more players with the best combination of cards as the winners.
`prize`	Split the prize pool among the winners and recalculate their chips. If more than one players win the game, the pot is split to each of them equally.
`show low`	Show all the hole cards of players and pick out the players with the lowest combination of cards as the winners.
`show high`	Show all the hole cards of players and pick out the players with the highest combination of cards as the winners.
`show high low`	Show all the hole cards of players and pick out the players who have the highest and the lowest combinations of cards as the winners.
`prize high low`	Split the prize pool equally, and one portion for the winners of the highest combination of cards, and the other portion for the winners of the lowest combination of cards.
`get straight`	Pick out the 'straight' from given cards, if it exists. If there are more than one, pick out the greater one.
`get pair`	Pick out the 'pair' from given cards, if it exists. If there are more than one, pick out the greater one.
`get two pair`	Pick out the 'two pair' from given cards, if it exists. If there are more than one, pick out the greater one.
`get 3 of a kind`	Pick out the 'three of a kind' from given cards, if it exists. If there are more than one, pick out the greater one.
`get 4 of a kind`	Pick out the 'four of a kind' from given cards, if it exists. If there are more than one, pick out the greater one.
`get flush`	Pick out the 'flush' from given cards, if it exists. If there are more than one, pick out the greater one.
`get full house`	Pick out the 'full house' from given cards, if it exists. If there are more than one, pick out the greater one.
`rank low high`	Rank the given cards from low to high.
`rank high low`	Rank the given cards from high to low.
`low suit`	Pick out the cards of distinct suits. If there are more than one of the same suit, choose the smaller one.

Continued on next page

Table 10: Composition of the core set. (Continued)

`high suit`	Pick out the cards of distinct suits. If there are more than one of the same suit, choose the greater one.
`highest x`	Choose the top x highest-ranking card from given cards.
`lowest x`	Choose the top x lowest-ranking cards from given cards.
`total bonus`	Add up all players' chips as the prize pool, which is the total bonus.
`bonus for x`	Average the total bonus or prize pool to x winners.
`add x chips`	Add x to player a's bet.
`drop x chips`	Take x from player a's bet.
`bet check`	Define a user operation called 'Check': Do nothing, only when the bet already matches the highest bet. This is specified by the user: 'pa: Check'.
`bet call`	Define a user operation called 'Call': Match the amount of the highest bet. This is specified by the user: 'pa: Call'.
`bet raise to x`	Define a user operation called 'Raise': Increase the bet to a higher bar. This is specified by the user: 'pa: Raise to x'.
`bet fold`	Define a user operation called 'Fold': Discard the hole cards and forfeit all chips already committed to the pot. This is specified by the user: 'pa: Fold'.
`show high x`	Show all the hole cards of players, and only a combination of x hole cards can be considered. Pick out the players with the highest combination of x cards as the winners.
`show low x`	Show all the hole cards of players, and only a combination of x hole cards can be considered. Pick out the players with the lowest combination of x cards as the winners.
`highest no pair`	Select the highest-ranking card with no pair.
`lowest no pair`	Select the lowest-ranking card with no pair.
`group suits`	Group the given cards by their suits.
`rank`	Rank the given cards.
`get all`	List all possible combination of cards from given cards. If there are one more for each combination, choose the greatest one.
`len`	Get the number of cards.

I In-Context Learning

We place three in-context samples for "All-in" in the game script. Note that this input is exclusively for samples of `bet`.

Table 11: Few-shot samples for in-context learning.

`Game script`

Continued on next page

Table 11: Few-shot samples for in-context learning. (Continued)

Let's be a Poker engine for Texas hold'em.
There are 5 players in the game. The minimum and maximum bet of the game is 2 and 1000.
Number: 2<3<4<5<6<7<8<9<10<11<12<1
Suit: H D C S
Hand: High Card<Pair<Two Pair<3 of a Kind<Flush<Full House<4 of a Kind<Straight Flush
Flow: start->blind->deal2->bet->flop3->bet->flop1->bet->flop1->bet->show->prize
Define a new player operation in the phase of bet `All-in`. The player puts all his remaining chips into the pot, and is no longer able to make further bet during the game.

For example:
Input 1:
|chip|p1: 10/990|p2: 0/1000|p3: 0/1000|p4: 5/995|p5: 10/990|p6: 10/990
|message|engine|p5|It's your turn to bet.
|message|p5|engine|All-in.
Response 1:
|chip|p1: 10/990|p2: 1000/0 (all-in)|p3: 0/1000|p4: 5/995|p5: 1000/0|p6: 10/990
|message|engine|p6|It's your turn to bet.

Input 2:
|chip|p1: 0/1000|p2: 0/1000|p3: 0/1000|p4: 0/1000|p5: 0/1000|p6: 0/1000|p7: 0/1000
|message|engine|p6|It's your turn to bet.
|message|p6|engine|All-in
Response 2:
|chip|p1: 10/990|p2: 10/990|p3: 10/990|p4: 0/1000|p5: 10/990|p6: 1000/0 (all-in)|p7: 0/1000
|message|engine|p7|It's your turn to bet.

Input 3:
|chip|p1: 24/976|p2: 30/970|p3: 0/1000
|message|engine|p1|It's your turn to bet.
|message|p1|engine|All-in
Response 3:
|chip|p1: 1000/0 (all-in)|p2: 30/970|p3: 0/1000
|message|engine|p2|It's your turn to bet.
Input
|order|p1|p2|p3 (button)|p4 (small blind)|p5 (big blind)

Continued on next page

Table 11: Few-shot samples for in-context learning. (Continued)

|chip|p1: 10/990|p2: 0/1000|p3: 0/1000|p4: 5/995|p5: 10/990
|stack|C12|S1|H10|S2|S7|C4|S5|C6|D4|D12|C8|H12|C11|D7|H3|D3|D10|C1|H5|H2|H6...
|hole|p1|H8|D1|p2|S6|C5|p3|D8|S4|p4|S3|C10|p5|S12|D9
|start|blind|deal2
|message|engine|p2|It's your turn to bet.
|message|p2|engine|All-in.

Response

|order|p1|p2|p3 (button)|p4 (small blind)|p5 (big blind)
|chip|p1: 10/990|p2: 1000/0 (all-in)|p3: 0/1000
|stack|C12|S1|H10|S2|S7|C4|S5|C6|D4|D12|C8|H12|C11|D7|H3|D3|D10|C1|H5|H2|H6...
|hole|p1|H8|D1|p2|S6|C5|p3|D8|S4|p4|S3|C10|p5|S12|D9
|start|blind|deal2
|message|engine|p3|It's your turn to bet.