



# Cardiverse: Harnessing LLMs for Novel Card Game Prototyping

Danrui Li<sup>1</sup>, Sen Zhang<sup>1</sup>, Sam S. Sohn<sup>1</sup>, Kaidong Hu<sup>1</sup>, Muhammad Usman<sup>2</sup>, Mubbasir Kapadia<sup>1,3</sup>

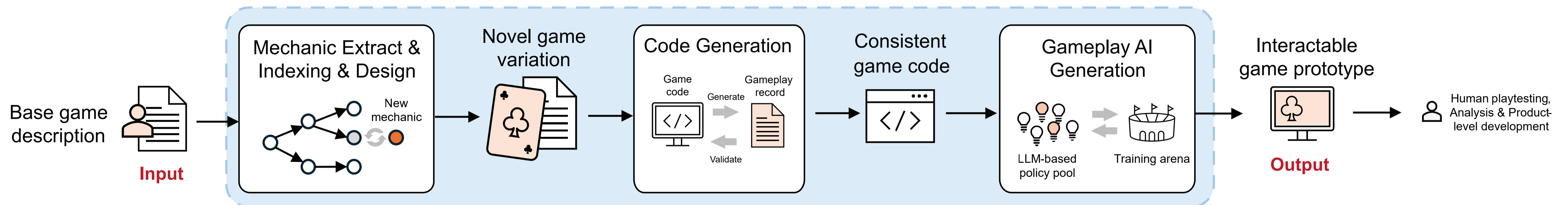
<sup>1</sup>Rutgers University <sup>2</sup>Ontario Tech University <sup>3</sup>Roblox

Find code repo and LLM card game arena at

<https://github.com/danruili/Cardiverse>



An LLM-based framework for card game prototyping.



## Challenge 1: Mechanics Novelty

### Problem

LLM may output game mechanics that is too similar to existing games.

### Solution

Use game mechanic graphs to enable informed, globally novel game mechanic design.

**LLM-based Mechanics Extraction**

The game ends → Empty hand → Laydown melds → Play cards that match rank

LLM Prompts: In this game (game\_description), we have extracted a logical chain (chain\_content). What game logic directly triggers Empty Hand?

LLM Response: To empty the hand, one has to either laydown melds or play cards that matches rank. (shown in JSON)

**Mechanics Graph Database**

The game ends (Root Node) → Keep smallest hand → Discard biggest card → Laydown melds → Play cards that match rank → Empty hand → Play a card with the same rank → Propose a replacement → Cluster and design: Play cards that differ suit

**Game input** → Mechanics Extraction → Mechanics graph

**Game variation** → Compose → Varied mechanics graph

**Metric: Quantiled Max Similarity at p**

- The pth percentile of the closest cosine similarity scores between the game variations and each game in the database or in all other variations.
- How close each new game is to those in the database

	p=0.75	p=0.50	p=0.25
<b>To database</b>			
Vanilla	88.7 ± 5.8 †	87.2 ± 6.3	83.8 ± 8.2
Prpt.Bdr.	88.3 ± 4.9 †	85.8 ± 4.5	81.1 ± 3.9 †
Ours	87.4 ± 4.5	84.8 ± 4.3	80.6 ± 3.7 †
<b>Within generated results</b>			
Vanilla	95.4 ± 0.8	92.0 ± 4.0	86.7 ± 5.9
Prpt.Bdr.	91.0 ± 2.1	85.5 ± 2.1 †	79.5 ± 2.5
Ours	89.6 ± 2.4	85.4 ± 2.8 †	81.3 ± 2.8

Ours yields lower similarity to known games (more originality) while preserving diversity among variations.

- Use graphs to represent game mechanic dependencies.
- Use LLMs to convert game descriptions into mechanic graphs.
- LLM expands nodes in BFS style.
- Group mechanics by meaning & depth using embeddings.
- LLM summarizes each cluster, adds creative variants.
- Given a game description, extract its mechanics, map them to clusters and find the ones that occur most frequently across the database.
- Replaces the frequent mechanics with newly designed ones from the same cluster

## Challenge 2: Code Consistency

### Problem

LLM-generated code may deviate from the game mechanics description text.

### Solution

Gameplay-reflective coding agent evaluates game consistency via gameplay records.

**Gameplay record**

Player 0 draw 2 cards from the deck. Deck size decreased by 1. Player 0 plays 8diamond to the table...

**Simulate** → **Validate & Refine**

**Game code**

```
def proceed(state, logger):
    ...
    state["deck_size"] -= 1
    logger.info("Deck size decreased by 1")
```

**Code edits**

```
state["deck_size"] -= 1
logger.info("Deck size decreased by 1")
state["deck_size"] -= 2
logger.info("Deck size decreased by 2")
```

Feed LLMs with gameplay records and game description to identify game rule inconsistencies.

Based on identified inconsistencies, use a code snippet library to guide LLMs make code refinements

	Succ ↑	Exec ↑	ECon ↑	PCon ↑
Ours	27/29	100.0 ± 0	0.55 ± 0.04	8.99 ± 1.56
-val	27/29	99.9 ± 3.9	0.56 ± 0.03 †	7.69 ± 1.96
-val ⊕	29/29	96.3 ± 9.4	0.56 ± 0.04	8.90 ± 1.74 †
<b>(a) Common Games</b>				
Ours	25/28	100.0 ± 0	0.52 ± 0.05	9.20 ± 1.08
-val	25/28	98.3 ± 4.6	0.52 ± 0.04	8.66 ± 1.60
-val ⊕	28/28	99.3 ± 2.0	0.54 ± 0.05	9.08 ± 1.29 †
<b>(b) Varied Games</b>				

- 100% executable game code.
- Achieve better consistency than ablated versions.

## Challenge 3: GameAI Scalability

### Problem

LLM-based agents are expensive to run and traditional RL requires long training time + strong hardware

### Solution

A LLM-based pipeline to create an ensemble of heuristic action-value functions, which doesn't require LLM calls in deployment.

**Game description** → **Propose strategies** → **Game strategies** → **Create code** → **Policy code pool** → **Select policy** → **Final gameplay AI**

**Strategy as text**

**Strategic Discarding**

Choose to discard cards that are least likely to be useful to other players based on previously observed discards and draws. Avoid discarding cards that could easily be used to complete an opponent's meld.

**Policy as code**

```
def q_func(state, action):
    ...
    if 'ace' in state['hand']:
        score += 1
    ...
    consecutive = []
    for card in state['hand']:
        ...
        score += len(consecutive)
    ...
    return score
```

**GameAI as heuristic function ensemble**

Game state dictionary

```
{ "public": { "deck_size": 18 },
  "players": [
    { "hand_size": 5, "meld": ... },
    { "hand_size": 8, "meld": ... },
    { "hand_size": 3, "meld": ... } ] }
```

Proposed action: Discard 4♦

1 Keep small hand → 0.2  
 2 Strategic discard → 0.0  
 3 Stop other player → 0.3  
**Action value = 0.5**

Our method beat CoT, ReAct, Reflexion, and BeliefAgent in the win rate advantage to random players.

Given the text description of card game, use LLMs to propose a set of game strategies.

Use LLMs to convert game strategies into Python function code, returning a score given the current game state and proposed action.

Iteratively optimize the selection of functions, evaluating each heuristic's win rate, adding the most effective ones step by step.