OPEN ROLE-PLAYING WITH DELTA-ENGINE

Anonymous authors
Paper under double-blind review

ABSTRACT

Game roles can be reflections of personas from a parallel world. In this paper, we propose a new style of game-play to bridge self-expression and role-playing: *open role-playing games (ORPGs)*, where players are given the autonomy to craft and embody their unique characters in the game world. Our vision is that, in the real world, we are individually similar when we are born, but we grow into unique ones as a result of the strongly different choices we make afterward. Therefore, in an ORPG, we empower players with freedom to decide their own growing curves through natural language inputs, ultimately becoming unique characters. To technically do this, we propose a special engine called *Delta-Engine*. This engine is not a traditional game engine used for game development, but serves as an in-game module to provide new game-play experiences. A delta-engine consists of two components, a base engine and a neural proxy. The base engine programs the prototype of the character as well as the foundational settings of the game; the neural proxy is an LLM, which realizes the character growth by generating new code snippets on the base engine incrementally. In this paper, we self-develop a specific ORPG based on delta-engines. It is adapted from the popular animated series "Pokémon". We present our efforts in generating out-of-domain and interesting role data in the development process as well as accessing the performance of a delta-engine. While the empirical results in this work are specific, we aim for them to provide general insights for future games. [1]

1 INTRODUCTION

The virtual world, often more idealistic than reality, presenting a utopian escape and an alternative life, has captivated human imagination for decades. Films like "Free Guy" and "Ready Player One" have presented this vision for us. Role-playing games (RPGs) offer players the opportunity to step into a well-designed character and enjoy its growth in a virtual game world, e.g. an Egyptian guard in "Assassin's Creed", an American West bounty hunter in "Red Dead Redemption". However, conventional RPGs come with inherent limitations, where players are bound to some pre-scripted characters. As time passes, they may find themselves merely acting out someone else, without any personal connection. Rather, players desire to become another version of themselves in a parallel world. To fulfill such deepest desire for autonomy and self-expression, this paper introduces the concept of *open role-playing games (ORPGs)*, which allow players to craft their own identities, attributes, and powers, etc.

How ORPGs are played to make each player's character distinguished In reality, while we are individually similar at birth, we make different choices (whether actively or passively) as we grow up afterward, which shape us into totally different people. This vision underpins the primary feature of ORPGs: players are given the autonomy to manipulate the growth of their characters and become truly unique ones. We notice that current RPGs also offer some extent of autonomy by providing players with different options when their characters grow. However, it is very hard to exhaust every possible option and fulfill the desire of every player. In this paper, we consider a generalized situation, where players are able to direct the growth with free-form natural language descriptions, e.g. "Let me learn a talent to burn the enemy". Such broad semantic space of natural language unlocks an unprecedented openness over traditional RPGs. However, the openness in ORPGs do not mean the players can become anybody or anything without constraint. All of the attempts should be contextualized within the specific game world, e.g. adhering to the physical laws and established worldview.

[1] Code, data, and demonstration are in our supplementary materials.

How ORPGs are technically implemented As aforementioned, the character growth in ORPGs is driven by natural language, which is difficult to interpret for existing game systems. We thus propose *Delta-Engine*. It is a neural engine (Wu et al., 2024a) incorporating a large language model (LLM) (Brown et al., 2020; OpenAI, 2023; Touvron et al., 2023; Jiang et al., 2023; Team et al., 2024). LLMs can serve as a powerful non-linear function to transfer natural language instructions into some kinds of outputs, such as the engine's code. Specifically, a delta-engine is composed of two components: a base engine and a neural proxy. The base engine is the initial coding of the character. The neural proxy is an LLM, enabling the character growth by generating new code snippets that expand the base engine. A delta-engine is different from traditional game engines (e.g. Unity, UE), which serve as a platform for game development, but serves as an embedded module within the game system. It would bring an entirely novel game-play experience, where the code of a player's character is personalized and dynamically generated.

To materialize our concept, we have developed a tangible ORPG *Free Pokémon* based on delta-engines, which will be open-sourced. The characters in the game are inspired from the popular Pokémon animated series[2]. In the game, players are born with a common pokemon template. From there, they have the freedom to grow up, learning their desired talents, described through natural language, being a unique pokemon to their tastes. An ORPG is data-driven. Therefore, data acts as a significant element in the development process, for aligning the neural proxy to enhance its adaptability. However, the data needed can be very domain-specific. Given the high cost of manually annotated data, this paper explores a collaborative design approach that leverages both human expertise and LLMs to generate high-quality data.

2 Related Work

AI-driven RPGs belong to a profound field of study that spans a series of sub-topics, e.g. role-playing (Shanahan et al., 2023; Värtinen et al., 2024; Wang et al., 2024), narrative (Wu et al., 2024c; Zhao et al., 2024a;b), world building (Bruce et al., 2024; Wang et al., 2023b), data creation (Wang et al., 2023c; Rezwana & Maher, 2023). This paper focuses on the form of the virtual world, i.e. to evolve, and proposes a special engine to enable such evolution. The basic logic of our world engine is the instruction-driven game engine (Wu et al., 2024b), a neural engine incorporating LLMs as integral components. Our engine can be triggered by a high-level evolving instruction to generate new code snippets. A relevant recent work is GameNGen (Valevski et al., 2024), a real-time game engine based on diffusion models. As opposed to GameNGen, which renders the content directly from prompts, our delta-engine generates the executable code and embed it into the base engine. The eventual rendering and operation is still done by the backbone engine, which is made by code.

The chosen playground in our work belongs to role-play games (RPGs), a genre that has seen significant advancements in recent years due to the integration of LLMs (Wu et al., 2024c; OpenAI, 2023; Touvron et al., 2023; Jiang et al., 2023; Yang et al., 2023). As opposed to these efforts, our work enhances the player experiences by offering biodiversity of virtual roles. Any role can become a truly unique one through exclusive evolution.

We are not the first to choose "Pokémon" as the topic in the research. A most recent work is Pokéllmon (Hu et al., 2024), but they do an orthogonal job from us. Pokéllmon is an LLM-based framework for powerful battle strategies based on existing pokemon characters. Free Pokémon is a novel game genre, allowing players to generate their own pokemon characters (Butler et al., 2017). In addition, our work does not focus on generating visual assets of new pokemon characters (Liapis, 2018; Geissler et al., 2020). We notice that it is also an interesting line for further developing ORPGs.

Procedural content generation (PCG) (Shaker et al., 2016; Smith et al., 2011; Summerville et al., 2018) can be another relevant line of work to ORPGs. Practically, the player's behavior won't directly affect the delta-engine, but rather go through a procedure, which eventually decides the prompt to the neural proxy. In this paper, we do not focus on the design of the evolving procedure but only study the naked delta-engine.

There is a large body of work studying AI players across various game domains, e.g. Atari (Mnih et al., 2013), Minecraft (Fan et al., 2022; Wang et al., 2023a), StarCraft, (Vinyals et al., 2019), Werewolf (Xu et al., 2023), SIMA (Team et al., 2024), CRADLE (Tan et al., 2024).

[2] https://www.pokemon.com/us

3 DELTA-ENGINE

Base Engine A base engine is the initial state of the delta-engine. It depicts the prototype of the virtual world, typically a mass of objects with associated methods and basic utilities.

As any object (e.g. environment, individual) grows, it acquires new properties. As a result, its associated engine is given new code. Considering an individual born with a blank template, its initial engine may be several lines of code supporting its only walking ability. As it grows stronger and learns to run and even fly, its codebase will be updated and expanded to reflect these new properties.

Neural Proxy The neural proxy is a neural wrapper around the base engine, which scales the base engine by producing new code. In our paper, it is a large language model (LLM), particularly one of those that are additionally pre-trained on code, e.g. CodeLLaMA (Rozière et al., 2023), CodeGemma (Mesnard et al., 2024).

We denote all the objects and methods of the engine at some moment as a state. The LLM proxy seeks to predict the new state moment-by-moment. To make this process efficient, we ensure that the proxy always generates the incremental code on top of the current engine state, either adding new features or overloading existing ones.

Incremental Prediction Given an input and the current engine state, a delta-engine seeks to predict the incremental value. This idea can be formalized as:

$$\Delta y_t = \mathcal{F}(y_{t-1}, x_t) \tag{1}$$

where $\mathcal{F}$ is the neural proxy, x_t is the input, y_{t-1} is the current engine state, and Δy_t is the incremental value of y_{t-1} and y_t. The initial state y_0 is the base engine.

y_t can be obtained by merging Δy_t and y_{t-1}:

$$y_t = m(\Delta y_t, y_{t-1}) \tag{2}$$

where m is the merge function.

Retrieval y_0 can be super large for a complicated virtual world and y_{t-1} will also become more and more as it evolves, negatively impacting the engine's scalability. We notice that evolution, both in nature and in virtual worlds, tends to occur gradually, which means each evolution step of an object is relevant to only a small fraction of the current engine. Therefore, for each prediction, we retrieve the relevant parts of the engine dynamically, denoted as $\widetilde{y}_{t-1}$, to replace the entire y_{t-1} as the reference:

$$\Delta y_t = \mathcal{F}(\widetilde{y}_{t-1}, x_t). \tag{3}$$

$\widetilde{y}_{t-1}$ is a sparse version of y_{t-1}, which is the key to the delta-engine's scalability to very long turns.

We have the neural proxy determine the entries to retrieve itself (Yu et al., 2023). Concretely, it takes a pre-step, predicting which parts of the engine are essential for scale. To do this, we index all methods within the engine using their names and prompt the neural proxy with the skeleton overview of the engine, which only keeps the structure and method names and skips the detailed implementation. Then, we extract the implementation of the methods from the engine according to the method names.

We will illustrate a concrete case of incremental prediction in the next section.

4 PLAYGROUND: FREE POKÉMON

Free Pokémon is developed based on delta-engines, which allows open-ended evolution for pokemon roles. Users can write their ideas in natural language, which directly manipulates the evolution step of the roles. As a result of that, the pokemon roles are able to acquire customized abilities and moves widely different from those in official games.

Figure 1 demonstrates the Free Pokémon system. It consists of two kinds of engines, role engine and battle engine. Every single pokemon role corresponds to a role engine, which is a delta-engine. It scales at each evolution step. The battle engine is responsible to host the battles between different pokemon roles. First, we can initialize the pokemon role by providing some basic settings, e.g.

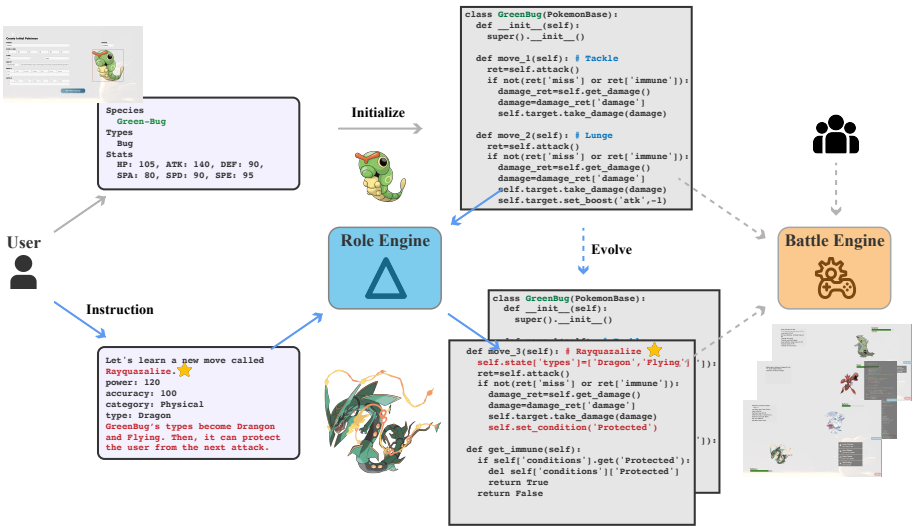

Figure 1: Free Pokémon system. Please see our supplementary materials for web demonstration.

species, types, and stats, which then will be transformed into a json format. In Free Pokémon, the initial states of all roles are almost the same, which is done by rules. Here, the user crafts a pokemon "Green-Bug" of Bug type. Then, its **role code** is initialized, with moves Tackle and Lundge. Specifically, the role is instantiated as a subclass `GreenBug` of `PokemonBase`. The `move_1`, and `move_2` correspond to its two moves respectively.

The blue stream in Figure 1 demonstrates the role's evolving process. This user provides a natural language description of his desire, letting his role learn a new move "Rayquazalize", whose secondary effect is to switch types and protect it from the next attack. This instruction will be sent to the role engine and trigger its scaling. As a result, it generates two new methods under the subclass. This scaling process will repeat for each evolution step.

Free Pokémon is an open-sourced playground for researchers interested in delta-engines. To facilitate research, the delta-engine is exposed directly to the researchers/users. They are able to manipulate the delta-engine by issuing any instructions, inputting anything they like. For example, one can craft a "Thanos" pokemon that owns super powerful moves to beat any other pokemons in one turn; even intentionally make problematic instructions to access the engine's behavior.

In Figure 2, we showcase the template we use for incremental prediction in Free Pokémon. In the first step, we prompt the neural proxy to decide the entries by providing a structural overview of the engine. Here, the proxy decides to retrieve two methods, `get_power` and `set_boost`. In the second step, we retrieve the implementation of these two methods as the reference for the neural proxy and prompt it to evolve the role. Eventually, it generates the incremental value of the engine as the response. Specifically, in the response, we develop a decorator "Increment" to merge the new code into the engine.

5 Training Data Generation

The delta-engine transfers the development process of the system to a hybrid of programming and data engineering. Developers are tasked to craft a sufficient amount of data to align the neural proxy. This process is labor-intensive. Even experienced professional designers can't always come up with fresh and innovative ideas. A recent philosophy is to synthesize pseudo data using powerful LLMs as generators (Wang et al., 2023c; Wu et al., 2024c). However, the synthetic data is somewhat low-quality and there can be significant and unknown biases within it (Shumailov et al., 2023). Instead, we adopt a human and AI co-design process, where LLMs are harnessed as assistant designers, working collaboratively with human designers. We first discuss two major demands of generated data.

```
## Input for Retrieve
You are a game programmer for Pokemon. I
will give you a structural overview of
an engine implementation of a pokemon
role and a natural language instruction.
You are tasked to retrieve the entries
you need to further implement the
instruction.

Overview:
PokemonBase
-  get_power
-  get_stat
-  attack
-  get_damage
-  set_boost
-  ...

Evolve instruction:
{instruction}

## Response
get_power
set_boost
```

```
## Input for Evolve
You are a game programmer for Pokemon. I will give
you a python implementation of a pokemon role and
a natural language instruction. You are tasked to
follow the instruction to evolve the pokemon role.
You can overload methods or add new ones.

Current implementation:
```python
{state}
```

Evolve instruction:
{instruction}

Response
```python
@Increment(Pikachu,'_move_3')
def value():
  return ('Quick Attack',40,100,
  'Physical','Normal',1,['contact'])

@Increment(Pikachu)
def move_3(self): # Quick Attack
  ret=self.attack()
  if not (ret['miss'] or ret['immune']):
    damage_ret=self.get_damage()
    damage=damage_ret['damage']
    self.target.take_damage(damage)
```
```

Figure 2: Input-output template for incremental prediction in Free Pokémon. The engine is implemented using Python. For brevity, we omit some elements: the engine state y_{t-1}, instruction x_t.

Being Novel Players are highly creative. For example, they won't be satisfied with similar content for long; they keep discovering novel and imaginative elements in the virtual world. Therefore, it is crucial for the delta-engine to scale to a broad range of novelty. However, we find that LLMs are not good at creating novel content based on given instances. Rather, they lean to combine existing content, such as merging two talents into a new one; or make superficial modifications, such as transforming a regular dog to a bigger dog. Such secondary data no longer enhances scalability, the emergence of which necessitates a leap into out-of-domain content.

Being Interesting Interestingness further aligns the delta-engine to the player base. Our demand is to refine the data design process by picking out the interesting portion from large amount of data. As opposed to novelty, which can be straightforwardly measured using similarity scores, however, quantifying interestingness has always been a challenging task (Nelson & Mateas, 2007; Todd et al., 2024). It is highly subjective and lacks a precise definition, making it difficult to devise instructions for LLMs to assess.

5.1 Prototypes Enhanced Imagination

We conjecture that LLMs lack or even do not have imagination; their creative outputs are still guided by the prompts they receive. However, the naive prompts e.g. "please use your imagination" fail to offer useful clues to inspire the LLM's imagination. To address this, we propose to leverage an explicit prototype, a descriptive paragraph of an entity or scene, as the imaginative foundation. It facilitates the generation of novel content by providing a concrete reference point.

Figure 3 illustrates the idea. For example, we seek to use *Tyrannosaurus* as the prototype to design a pokemon role. We retrieve the corresponding description from Wikipedia and prompt the LLM generator. The result is a novel pokemon characterized by stronger bite power, aligning with the notable feature of Tyrannosaurus. In addition to real-world entities, prototypes can also come from fictional sources. In our project, we retrieve the animals from Wikipedia, e.g. *Tyrannosaurus*, *Smilodon*, *Sperm Whale*; also retrieve the virtual creatures from "Monster Hunter"[3], a popular action video game. The distinction between the two sources is that the super-natural creatures in Monster Hunter typically lead to higher novelty of the pokemon roles designed upon them. However, these roles may go too far, increasing the likelihood of grammatical errors within the generated code, which we will discuss below.

[3] https://www.monsterhunter.com

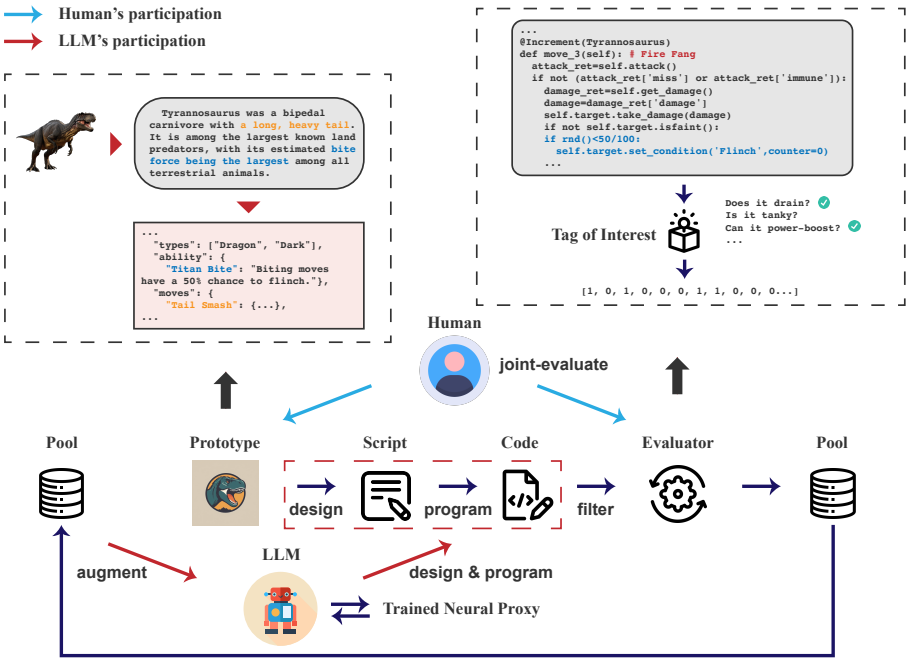

Figure 3: Human and AI design (co-design). At the top left, we illustrate the process we leverage prototypes to enhance the LLM's design. We align the descriptions of the prototype and its associated design result using colors.

5.2 Tags of Interest

We hypothesize that interestingness is an accumulation of potential factors that may pique the users' interest (Althöfer, 2010). This implies that the more potential factors there are, the more likely the users will find the content interesting. Therefore, we introduce an interestingness evaluator based on these factors, which we call *Tags of Interest (ToI)*. We then need a tagger to label them out given the instance.

Firstly, we establish a set of ToI. Since they vary significantly across different scenarios, this is a heuristic process. We can construct a one-dimensional "interestingness vector", where each tag is represented as one bit. For example, if a pokemon role has the ability to boost its power, the corresponding bit of this tag will be set to 1; otherwise, set to 0. We use a rule-based tagger to mine the potential tags of a role from its role code. For example, if a pokemon role can boost its power, it will inevitably overload the method `get_power`. Based on the interestingness vector, we set a threshold; if the magnitude of the vector does not reach the threshold, the sample will be filtered out.

5.3 Human and AI Co-Design

Figure 3 illustrates our co-design process with the dual participation of human and AI (LLM) designers to generate the training data we need. In this process, we seek to generate two parts of data, **role script** and **role code**. The former is a natural language json script that details the pokemon role. We use a script-code pair to identify a role. Eventually, we split each role into several states as the training samples for doing incremental prediction.

Concretely, we initialize the sampling pool with 20 manually-crafted seed instances of script-code pairs. The human designer first determines the prototype and prompts the LLM designer to generate a novel role script based on the prototype. Then, the LLM designer is prompted to program the role script into the role code. More specifically, we sample 5 instances from the sampling pool to augment the LLM's coding, while only sampling 1 instance to augment its designing. A key observation is that the in-context instances of other role scripts may bias the effect of the provided prototype, incurring low creativity of the response. On the other hand, in-context instances act as useful references for the

LLM to generate high-quality role code since the programming step does not rely on creativity but accuracy. The LLM designer we use is either of GPT4 or Claude3. The newly designed script-code pair will be sent to the evaluator, which is a joint process with both rule-based and manual strategies. First, code that fails to compile or introduces new methods yet without calling them will be filtered out. Second, code that fails to pass the interestingness threshold will also be discarded. After the rule-based filtering, the human designer makes the final check on the script and code. Eventually, we place the new instance into the sampling pool ready for the next cycle of design.

An important trick is that, we replace the third-party LLM designers (GPT4/Claude3) with one of the trained neural proxy in the middle of the design process. We find them, yet powerful, still struggle with the nuanced requirements of the programming problem in our project, providing low-accuracy responses, while the trained model can tackle much better.

The incorporation of AI greatly accelerates the creative process of human designers (Rezwana & Maher, 2023), creating high-quality data. In Figure 3, we observe that human designers mainly act as a prototype designer and a joint evaluator to refine the eventual instances.

6 EXPERIMENT

This section reports our experiments. The results are based on our chosen domain Free Pokémon. Nonetheless, if the proposed methods effectively work on our domain, it is highly promising to generalize them in the future.

6.1 BASIC SETTING

To access the quality of the co-designed data, we prepare another set of data of the same size purely synthesized by Claude3 . Specifically, the synthetic data is generated using a similar pipeline as in Figure 3. The difference is that we automatically sample the official pokemon roles as the prototypes, while canceling the manual evaluation step, since these two steps necessitate human participation. We show the data statistics in the upper half of Table 1.

On the other hand, we prepare two sets of test data, corresponding to **easy** and **hard**. The easy-level data comprises 19 existing pokemon roles, all of which have appeared in official pokemon games. We sample them from the internet. This set of data is easier because the majority of roles in the training data, including purely synthesized and co-designed, inevitably share similarity with the existing ones. Their distributions are closer as a result. To deeply access the scalability of the engine, in addition, we invite 10 volunteers to manually craft the hard-level data. All of them are not only experienced in playing pokemon games, but also have a wealth of experience with a wide range of games, greatly allowing them to design novel pokemon roles. Eventually, we obtain 16 original role scripts. We manually program them and obtain the ground truth role code. Beyond originality, volunteers are asked to craft more moves and abilities for one role, which helps us to better evaluate the scalability. From Table 1, we observe that the number of evolution steps and sentence length of hard-level data is much more than those of easy-level data.

We fine-tune the CodeGemma-7b model (Mesnard et al., 2024)[4]. CodeGemma is a code LLM that is additionally pre-trained on a large number of code corpora. We train each model using LoRA (Hu et al., 2022) with $r = 8$, $\alpha = 32$, learning rate 1.5e-4, and batch size 4 for 5 epochs.

We report two scores.

Exe%: We calculate the success rate of executing the role code on top of the engine. Specifically, we randomly synthesize 100 roles as imaginary opponents and have the role under test to battle with them, choosing a random action each time. One success will be counted if all actions are executed successfully against all opponents.

Acc%: We further verify the accuracy/correctness of the role code. This step is done by GPT4, which is prompted to compare two code snippets. Note that we calculate the correctness only among the successfully executed code.

[4] https://huggingface.co/google/codegemma-7b-it

Table 1: Upper: Dataset statistics, in order: the number of roles we created, the number of samples, the average number of evolution steps of each role, and the average sentence length. To calculate the sentence length, we use the CodeGemma tokenizer. Lower: Results on different test sets. ✓ indicates the 100% score. "Retr." refers to the retrieval technique.

Statistic	**Roles**	**Samples**	**#Evolves**	**#Length**
SY. TRAIN	167	500	3.0	1197.8
CO. TRAIN	175	502	2.9	1167.3
EASY TEST	19	43	2.3	997.2
HARD TEST	16	87	**5.4**	**1841.6**

	Easy		**Hard**	
Performance	Exe	Acc	Exe	Acc
CODEGEMMA W. SY.	95.3	86.0	86.2	58.6
CODEGEMMA W. CO.	✓	95.3	90.8	83.9
CODEGEMMA W. CO. W. RETR.	✓	✓	**92.0**	**89.7**

6.2 MAIN RESULTS

From the lower half of Table 1, we find that the two models trained on synthetic data and co-designed data (Sy. & Co.) perform comparatively on the easy test set. This is due to the closer gap between training and test data in this scenario. More specifically, the co-designed data by both humans and AI performs slightly better. The resultant model and its retrieval-augmented version achieves a full Exe rate, and the latter also achieves a full Acc rate.

On the other hand, the hard test data delivers a large distribution gap from the training data, leading to noticeable performance drop across all three model counterparts. More importantly, the gap between Exe and Acc becomes more pronounced. The elevated Exe rate indicates that the trained model is inclined to respond with executable code even if the input role is unfamiliar. However, the accuracy of code is much harder to fulfill. We find that the model trained only on synthetic data is notably weaker. This is due to the fact that the synthetic data is too limited and doesn't provide useful signals for out-of-domain generalization. In contrast, the co-design process produces high-quality data with out-of-domain signals, which significantly generalizes the model, improving the Acc rate from 58.6 to 83.9. Furthermore, we find that the retrieval technique also shows its positive impact, further improving Acc to 89.7.

In addition to out-of-domain generalizability, the delta-engine's scalability includes its scaling performance through long evolution steps. To do this, we conduct another experiment where we continuously scale the delta-engine. Specifically, we randomly sample abilities and moves from the existing database and repeatedly prompt the neural proxy to scale, until it gives a non-executable response. The result will be a "super patchwork" pokemon role. We repeat this process 100 times. Figure 4 shows two histograms, where we demonstrate the scaling performance of the engine through the evolution steps and the engine size. The engine size refers to the number of tokens of its current code. Intuitively, we observe that as the evolution increments, the performance exhibits a pronounced degradation. More specifically, as the evolution accumulates to 20 steps, only half of the cases give an executable response. A similar trend can be seen as the engine size accumulates to 5000. Note that, the length limit of the CodeGemma model is 8192. However, we find that the introduction of the retrieval technique brings a nice scalability in the face length increasing. The resultant model maintains a nice performance up to 30 evolution steps. This is because the retrieved engine state is much smaller compared to the entire engine. We illustrate a case on the right side of Figure 4. The model only retrieves the `type_change` method from the engine as the context for the following incremental prediction.

So far, we haven't explore larger-sized LLMs, though they can be promisingly stronger.

6.3 DATA ANALYSIS

To further investigate why co-designed data outperforms synthetic data and the distinction between easy and hard test data, we take a closer look into the underlying data distribution. Therefore, we visualize the data points of pokemon roles from two distinct views in Figure 5. Specifically,

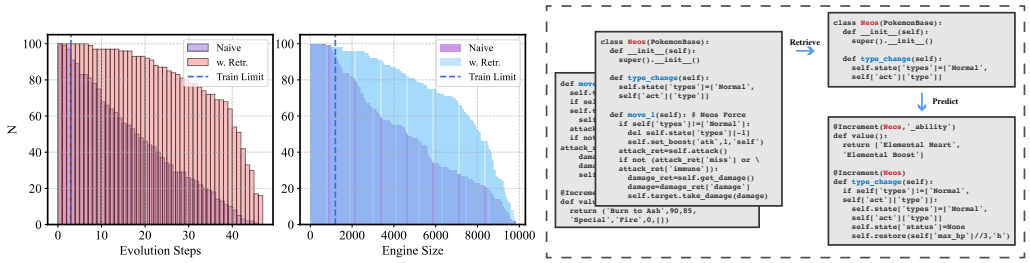

Figure 4: Histograms of 100 sampling. We highlight the number of evolution steps in the training data as a baseline. On the right, we show a concrete case of the retrieval process.

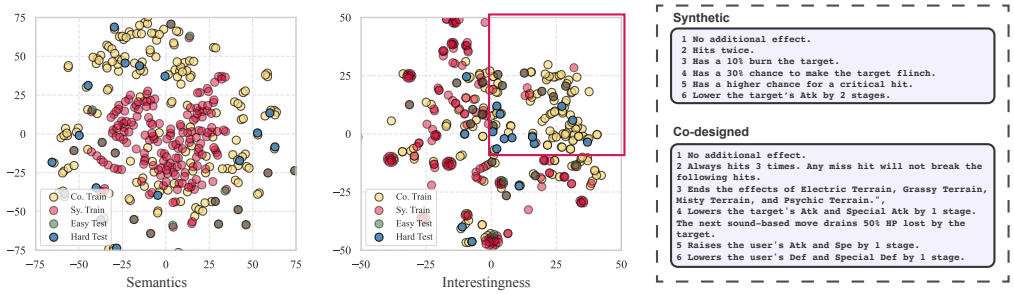

Figure 5: Comparison of the roles crafted by different methods and a concrete case on the right. We visualize them from the semantics and interestingness space.

we apply the sentence embedding model[5] to encode the role descriptions into vectors and obtain the interestingness vectors based on ToI. Then, we apply t-SNE to project all vectors into a two-dimensional space.

From the left side of Figure 5, we observe that the co-designed data points nearly encompass all synthetic data points, with the distribution of the latter exhibiting more converged. It highlights the fact that the co-designed outweighs the synthetic one in terms of semantic diversity, thus enhancing the training. We show a case on the right side. They belong to the same pokemon role yet are generated by different methods. We find that the synthetic data easily falls into an identical pattern, while the co-designed one exhibits great diversity. We find a quite different vision when we segment the data from interestingness. A similar observation is apparent that the co-designed data points continue to cover almost all synthetic ones. In particular, we notice that in the upper right, the area we have highlighted with a red box, there is a blind spot of the synthetic data. It means that the model trained solely on synthetic data, fails to capture meaningful signals from the test data in this area. Furthermore, we observe that most of the hard test data points, which are crafted by humans, are distributed in this area.

7 Conclusion

This paper concentrates on the evolving nature of the virtual world. We model this by proposing the delta-engine. The experiments are made on our self-developed playground Free Pokémon. This work is the initial attempt into the study, opening up a wealth of valuable topics for future research, e.g. developing a fully realized virtual world system, studying better training techniques to align the neural proxy, addressing the safety concerns.

References

Ingo Althöfer. Automatic generation and evaluation of recombination games. *J. Int. Comput. Games Assoc.*, 33(4):215–216, 2010. doi: 10.3233/ICG-2010-33405. URL https://doi.org/10.

[5] https://huggingface.co/sentence-transformers/all-mpnet-base-v2

3233/ICG-2010-33405.

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.

Jake Bruce, Michael D. Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bechtle, Feryal M. P. Behbahani, Stephanie C. Y. Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott E. Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative interactive environments. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=bJbSbJskOS.

Eric Butler, Kristin Siu, and Alexander Zook. Program synthesis as a generative method. In Sebastian Deterding, Alessandro Canossa, Casper Harteveld, Jichen Zhu, and Miguel Sicart (eds.), *Proceedings of the International Conference on the Foundations of Digital Games, FDG 2017, Hyannis, MA, USA, August 14-17, 2017*, pp. 6:1–6:10. ACM, 2017. doi: 10.1145/3102071.3102076. URL https://doi.org/10.1145/3102071.3102076.

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.

Dominique Geissler, Elisa Nguyen, Daphne Theodorakopoulos, and Lorenzo Gatti. Pokérator - unveil your inner pokémon. In F. Amílcar Cardoso, Penousal Machado, Tony Veale, and João Miguel Cunha (eds.), *Proceedings of the Eleventh International Conference on Computational Creativity, ICCC 2020, Coimbra, Portugal, September 7-11, 2020*, pp. 500–503. Association for Computational Creativity (ACC), 2020. URL https://computationalcreativity.net/iccc20/papers/159-iccc20.pdf.

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.

Sihao Hu, Tiansheng Huang, and Ling Liu. Pokellmon: A human-parity agent for pokemon battles with large language models. *CoRR*, abs/2402.01118, 2024. doi: 10.48550/ARXIV.2402.01118. URL https://doi.org/10.48550/arXiv.2402.01118.

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.

Antonios Liapis. Recomposing the pokémon color palette. In Kevin Sim and Paul Kaufmann (eds.), *Applications of Evolutionary Computation - 21st International Conference, EvoApplications*

2018, Parma, Italy, April 4-6, 2018, Proceedings, volume 10784 of *Lecture Notes in Computer Science*, pp. 308–324. Springer, 2018. doi: 10.1007/978-3-319-77538-8_22. URL https://doi.org/10.1007/978-3-319-77538-8_22.

Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Cristian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, and et al. Gemma: Open models based on gemini research and technology. *CoRR*, abs/2403.08295, 2024. doi: 10.48550/ARXIV.2403.08295. URL https://doi.org/10.48550/arXiv.2403.08295.

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.

Mark J. Nelson and Michael Mateas. Towards automated game design. In Roberto Basili and Maria Teresa Pazienza (eds.), *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, 10th Congress of the Italian Association for Artificial Intelligence, Rome, Italy, September 10-13, 2007, Proceedings*, volume 4733 of *Lecture Notes in Computer Science*, pp. 626–637. Springer, 2007. doi: 10.1007/978-3-540-74782-6_54. URL https://doi.org/10.1007/978-3-540-74782-6_54.

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.

Jeba Rezwana and Mary Lou Maher. Designing creative AI partners with COFI: A framework for modeling interaction in human-ai co-creative systems. *ACM Trans. Comput. Hum. Interact.*, 30(5): 67:1–67:28, 2023. doi: 10.1145/3519026. URL https://doi.org/10.1145/3519026.

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.

Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games*. Computational Synthesis and Creative Systems. Springer, 2016. ISBN 978-3-319-42714-0. doi: 10.1007/978-3-319-42716-4. URL https://doi.org/10.1007/978-3-319-42716-4.

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.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross J. Anderson. The curse of recursion: Training on generated data makes models forget. *CoRR*, abs/2305.17493, 2023. doi: 10.48550/ARXIV.2305.17493. URL https://doi.org/10.48550/arXiv.2305.17493.

Gillian Smith, Elaine Gan, Alexei Othenin-Girard, and Jim Whitehead. Pcg-based game design: enabling new play experiences through procedural content generation. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games, PCGames '11, Bordeaux, France, June 28, 2011*, pp. 7:1–7:4. ACM, 2011. doi: 10.1145/2000919.2000926. URL https://doi.org/10.1145/2000919.2000926.

Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine

learning (PCGML). *IEEE Trans. Games*, 10(3):257–270, 2018. doi: 10.1109/TG.2018.2846639. URL https://doi.org/10.1109/TG.2018.2846639.

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. *CoRR*, abs/2403.03186, 2024. doi: 10.48550/ARXIV.2403.03186. URL https://doi.org/10.48550/arXiv.2403.03186.

SIMA Team, Maria Abi Raad, Arun Ahuja, Catarina Barros, Frederic Besse, Andrew Bolt, Adrian Bolton, Bethanie Brownfield, Gavin Buttimore, Max Cant, Sarah Chakera, Stephanie C. Y. Chan, Jeff Clune, Adrian Collister, Vikki Copeman, Alex Cullum, Ishita Dasgupta, Dario de Cesare, Julia Di Trapani, Yani Donchev, Emma Dunleavy, Martin Engelcke, Ryan Faulkner, Frankie Garcia, Charles Gbadamosi, Zhitao Gong, Lucy Gonzalez, Kshitij Gupta, Karol Gregor, Arne Olav Hallingstad, Tim Harley, Sam Haves, Felix Hill, Ed Hirst, Drew A. Hudson, Jony Hudson, Steph Hughes-Fitt, Danilo J. Rezende, Mimi Jasarevic, Laura Kampis, Nan Rosemary Ke, Thomas Keck, Junkyung Kim, Oscar Knagg, Kavya Kopparapu, Andrew K. Lampinen, Shane Legg, Alexander Lerchner, Marjorie Limont, Yulan Liu, Maria Loks-Thompson, Joseph Marino, Kathryn Martin Cussons, Loic Matthey, Siobhan Mcloughlin, Piermaria Mendolicchio, Hamza Merzic, Anna Mitenkova, Alexandre Moufarek, Valéria Oliveira, Yanko Gitahy Oliveira, Hannah Openshaw, Renke Pan, Aneesh Pappu, Alex Platonov, Ollie Purkiss, David P. Reichert, John Reid, Pierre Harvey Richemond, Tyson Roberts, Giles Ruscoe, Jaume Sanchez Elias, Tasha Sandars, Daniel P. Sawyer, Tim Scholtes, Guy Simmons, Daniel Slater, Hubert Soyer, Heiko Strathmann, Peter Stys, Allison C. Tam, Denis Teplyashin, Tayfun Terzi, Davide Vercelli, Bojan Vujatovic, Marcus Wainwright, Jane X. Wang, Zhengdong Wang, Daan Wierstra, Duncan Williams, Nathaniel Wong, Sarah York, and Nick Young. Scaling instructable agents across many simulated worlds. *CoRR*, abs/2404.10179, 2024. doi: 10.48550/ARXIV.2404.10179. URL https://doi.org/10.48550/arXiv.2404.10179.

Graham Todd, Alexander Padula, Matthew Stephenson, Éric Piette, Dennis JNJ Soemers, and Julian Togelius. Gavel: Generating games via evolution and language models. *arXiv preprint arXiv:2407.09388*, 2024.

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.

Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion models are real-time game engines. *CoRR*, abs/2408.14837, 2024. doi: 10.48550/ARXIV.2408.14837. URL https://doi.org/10.48550/arXiv.2408.14837.

Susanna Värtinen, Perttu Hämäläinen, and Christian Guckelsberger. Generating role-playing game quests with GPT language models. *IEEE Trans. Games*, 16(1):127–139, 2024. doi: 10.1109/TG.2022.3228480. URL https://doi.org/10.1109/TG.2022.3228480.

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, 2023a. doi: 10.48550/ARXIV.2305.16291. URL https://doi.org/10.48550/arXiv.2305.16291.

Noah Wang, Z. y. Peng, Haoran Que, Jiaheng Liu, Wangchunshu Zhou, Yuhan Wu, Hongcheng Guo, Ruitong Gan, Zehao Ni, Jian Yang, Man Zhang, Zhaoxiang Zhang, Wanli Ouyang, Ke Xu, Wenhao Huang, Jie Fu, and Junran Peng. Rolellm: Benchmarking, eliciting, and enhancing role-playing abilities of large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pp. 14743–14777. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.878. URL https://doi.org/10.18653/v1/2024.findings-acl.878.

Ruoyao Wang, Graham Todd, Xingdi Yuan, Ziang Xiao, Marc-Alexandre Côté, and Peter A. Jansen. Bytesized32: A corpus and challenge task for generating task-specific world models expressed as text games. 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. 13455–13471. Association for Computational Linguistics, 2023b. doi: 10.18653/V1/2023.EMNLP-MAIN.830. URL https://doi.org/10.18653/v1/2023.emnlp-main.830.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pp. 13484–13508. Association for Computational Linguistics, 2023c. doi: 10.18653/V1/2023.ACL-LONG.754. URL https://doi.org/10.18653/v1/2023.acl-long.754.

Hongqiu Wu, Xingyuan Liu, Yan Wang, and Hai Zhao. Instruction-driven game engine: A poker case study. In Delia Irazu Hernandez Farias, Tom Hope, and Manling Li (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 507–519, Miami, Florida, USA, November 2024a. Association for Computational Linguistics. URL https://aclanthology.org/2024.emnlp-demo.51.

Hongqiu Wu, Yan Wang, Xingyuan Liu, Hai Zhao, and Min Zhang. Instruction-driven game engines on large language models. *CoRR*, abs/2404.00276, 2024b. doi: 10.48550/ARXIV.2404.00276. URL https://doi.org/10.48550/arXiv.2404.00276.

Weiqi Wu, Hongqiu Wu, Lai Jiang, Xingyuan Liu, Hai Zhao, and Min Zhang. From role-play to drama-interaction: An LLM solution. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pp. 3271–3290. Association for Computational Linguistics, 2024c. doi: 10.18653/V1/2024.FINDINGS-ACL.196. URL https://doi.org/10.18653/v1/2024.findings-acl.196.

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.

Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. Generate rather than retrieve: Large language models are strong context generators. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=fB0hRu9GZUS.

Runcong Zhao, Wenjia Zhang, Jiazheng Li, Lixing Zhu, Yanran Li, Yulan He, and Lin Gui. Narrativeplay: Interactive narrative understanding. In Nikolaos Aletras and Orphée De Clercq (eds.), *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2024 - System Demonstrations, St. Julians, Malta, March 17-22, 2024*, pp. 82–93. Association for Computational Linguistics, 2024a. URL https://aclanthology.org/2024.eacl-demo.10.

Runcong Zhao, Qinglin Zhu, Hainiu Xu, Jiazheng Li, Yuxiang Zhou, Yulan He, and Lin Gui. Large language models fall short: Understanding complex relationships in detective narratives. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pp. 7618–7638. Association for Computational Linguistics, 2024b. doi: 10.18653/V1/2024.FINDINGS-ACL.454. URL https://doi.org/10.18653/v1/2024.findings-acl.454.