Nobody Writes Docstring: Game Item Description Generation with Pretrained Language Models

Ning Wan **Hongyu Zhang** **Yuxuan Wang**
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
{ningwan, hz53, wangy49}@seas.upenn.edu

Abstract

Game item has always been an essential part of video games. The textual descriptions of these items often reveal crucial information about the game settings and background stories. They are particularly important for text-based adventure games since the text is the only medium for communication. However, writing detailed textual descriptions for hundreds of game items, such as cups and swords, can be tedious and cumbersome. Therefore, we explore the feasibility of automated description generation via large pre-trained language models in this work. We innovatively decompose the generation process into a "prototyping" phase and a "game rendering" phase to accomplish this task. Such a de-coupling brings about improved performance and engineering benefits. Finally, we establish a strong baseline that achieves high BERTScore on the LIGHT dataset, and the generated texts are also satisfactory by human standards.

1 Project Description

Nobody wants to write docstring. For games, it is time-consuming to manually write complete textual descriptions for each in-game item. What if we can build a better description generator? Recall the games you played; the item descriptions contains information particular to the game settings or the game-world settings For example, you know the item is from *Dark Souls* or *Don't Starve* simply by reading the text: even when they are not! Therefore, our work is more developer-oriented, particularly convenient for those who want to develop mods for their favorite games and conform to a consistent world setting or style of writing. So, in this project, we would like to revisit Homework Two, the description generation. We want a generator that takes an item title as input and generates a corresponding "gamified" item description. However, instead of building an end-to-end system as we have done in Homework Two, we observe that such a task is two-fold:

- Generate a basic description for a simplified item name that depicts the main properties of this item. For instance, a sword is made of metal, a pointy weapon.
- Render the utterance to sound like a game item when we condition it on the game category and full item name like *King Arthur's Sword*.

To sum, we want to formulate this task as a combination of description generation and conditioned style transfer problem.

2 Related Work

Our project leverages the recent advancement of large **Pre-trained Language Models** (PLM), e.g., GPT-3, T5, and BART (Brown et al., 2020; Raffel et al., 2019; Lewis et al., 2019). Further, it is shown that such PMLs can serve as knowledge bases, especially salient for simple factual knowledge (Hwang et al., 2020; Petroni et al., 2019). This inspires us of using large PLMs as base description generator.

Our formulation of game item description generation resembles the most to the **Text Style Transfer** (TST) task with some variation. TST aims to control certain attributes in the generated text, such as politeness, emotion, humor, and many others (Jin et al., 2020). It involves a source style and a target style. In our case, the source style is the simple item description, where as the target style is the gamified description. Specifically, given the base description, we can construct a parallel dataset where standard sequence-to-sequence models can be applied (Rao and Tetreault, 2018).

However, we need to keep in mind that the renderer functions not only as a style transferer but also as a domain-specific knowledge base. For example, to generate fantasy-game-related item descriptions, a renderer needs to have prior knowledge about terms like Mana, Goblin, or Holy Grail.

And we do not expect an out-of-the-box model has such knowledge pre-encoded (Gururangan et al., 2020). Therefore, the specific methods to solve this problem are under discussion.

3 Methods

3.1 Proposed Architecture

According to our observation, we can decompose the game-item description generation as a two-phase process: base form description generation and rendering. The two models are collated together to form a game item descriptor.

3.1.1 Base Generator

Prompting Baseline First of all, almost every game item has a "base form." Such base forms can be weapons, garments, or food. The base form of a game item reveals its essential properties. For instance, *Excalibur* is still a sword, a pointy weapon (possibly) made of a piece of metal. Therefore, intuitively, to generate a description for a game item, we need a particular mechanism to help create a piece of text for its base form in the first place. Such a machine needs to possess common sense knowledge about the properties of various base forms, which naturally leads to large PLMs or models trained upon knowledge bases. Formally, a generator is a function with the signature,

Listing 1: The generator model API. It should take a single item like 'sword' and generate some commonsense-based description for that item, e.g., 'a pointy piece of metal'. The `model` here is an abstraction, it can be a large PLM like GPT-3 or specialized knowledge-base model like COMET-BART

```
def generator(base_form: str) -> str:
    """Base Description Generator
            API Pseudo Code"""

    prompt = make_prompt_gen(base_form)
    base_desc = model_gen(prompt)
    return post_proc_gen(base_desc)
```

3.1.2 Game Renderer

Prompting Baseline Next, we believe that two extra pieces of information contribute to the "game flavor" of the description. Firstly, where do these items appear? For instance, a cup in a dungeon may associate with adjectives like "filthy" or "rusty," while a mug in a kitchen can be "clean" or "cute." Secondly, the full name of that item, e.g., *King Arthur Sword* instead of *Sword*. We know that this particular weapon is from *King Arthur's Legend*. So the corresponding descriptions should relate to terms like *Avalon*, *Knights of the Round Table*, or *Holy Grail*. We can see that it also requires the renderer to have another set of common sense knowledge. However, such knowledge is more game-specific or world-specific: it is not likely to have a machine gun in greek mythology. In other words, the renderer serves as an (inverse) adapter, and the engineering advantages brought by the disentanglement are apparent, i.e., we do not need to re-train the often-humongous base generator when switching to a different set of settings or game.

Listing 2: The renderer model API. It rephrase/diversify the item description conditioning on the `setting`, `full_name`, and pre-computed `base_desc`

```
def renderer(setting: str,
             full_name: str,
             base_desc: str) -> str:
    """Renderer API Pseudo Code"""

    prompt = make_prompt_ren(setting,
    full_name, base_desc)
    game_desc = model_ren(prompt)
    return post_proc_ren(game_desc)
```

3.1.3 Item Base Form Extractor

Another small but important piece of building block is the item base form extractor. Ideally, a user of our proposed system only needs to specify the full name of a game item, but a base generator only takes the **base form** as its input. Thus, we need a dedicated algorithm to extract/convert the game item to its base form. We currently explored two approaches. The first method is based on KeyBERT, which tries to find a subset of the tokens that best preserves the semantic information, measured by **cosine-similarity** between full name contextualized embedding and those of the sub-tokens'. However, the base form may not necessarily contained in the full name. So, another method we proposed is to, again, rely on large PLMs and text-generation. We experimented with generation using COMET-BART via *MadeOf* relation, and prompting-by-example with GPT-3. The GPT-3 model gives the best performance.

3.2 Model Development Procedure

We introduce both planned development procedure as well as preliminary experiments in this section.

3.2.1 Generator Construction

Prompting Baseline As discussed above, the base description generation relies on a large PLM that can serve as a knowledge base. Therefore, we develop our baseline model based on **GPT-3** via

few-shots prompting. We construct the prompt by concatenating ten "item: <ITEM>, description : <DESC>" pairs. An example is shown below,

```
prompt = f"""
item: Sunglasses,
description: All of our sung... \n
item: Scarf,
description: Feature Great quality... \n
                    ...
item: Shoe,
description: Kick back and relax... \n
item: {NEW_ITEM}
description: {TO_BE_COMPLETE}
"""
```

The item descriptions we used is acquired from the Amazon Review Dataset, which is discussed in section 4.

Fine Tuning We fine-tuned the data manually collected from Amazon Dataset and Wikipedia in order to make the generated description have advertising style with a bit formality. The tuning is comprised of the prompt with item name included and completion with description included. We try to make the generated description as much as possible to give out a through base description, so the maximum generated length keeps at the original upper limit in OpenAI.

3.2.2 Renderer Construction

Prompting Baseline Again, we use **GPT-3 prompting** for the proof-of-concept purpose. An example prompt is shown below,

```
prompt = f"""
item: Wood table
category: Tavern
base_desc: The table is made of wood ...
desc: The table is old and dusty ...
===\n
                    ...
item: {FULL_NAME}
category: {SETTING}
base_desc: {BASE_DESC}
desc: {TO_BE_COMPLETE}
"""
```

The base description is generated from pre-constructed base-desc-generator, and target description is taken from the LIGHT dataset, discussed in section 4.

Fine Tuning For the renderer, we fine-tuned the GTP-3 model by randomly choosing 8 items from each game category in LIGHT dataset. Specifically, for each item we chose, we record the room it was in, and then we could retrieve the corresponding game category. In addition, we also generate the base description of this item using the pre-constructed base-desc-generator discussed in section 3.2.1. Then, we construct the prompt by including the item name, game category, and its generated base description. The completion for this item is the full description we get from the LIGHT dataset, plus a stop token "===".

3.2.3 Special Model For Dark Souls

This model is targeted at generating dark-souls-styled item description. Due to the uniqueness of the explanations for most items in dark souls, it is separated from generalized version of docstring generation model. This model follows the same logic as the base and render generator part but with all the fine-tuned data set as crawled info from the official Dark Souls website.

4 Data

4.1 Amazon Review (Meta) Data

This dataset contains product reviews and metadata from Amazon, including 142.8 million reviews spanning from May 1996 to July 2014. This dataset includes reviews (ratings, text, helpfulness votes), product metadata (descriptions, category information, price, brand, and image features), and links (also viewed/also bought graphs) [1]. For this project, **we only use the metadata** which contains **item names** and corresponding **short descriptions**. But, we noticed that the dataset is noisy (for our purpose). For instance, it may include advertising text that is depicting the properties of the (base) item. Thus, we manually select a tiny subset of descriptions that are more relevant. Further, we plan to augment the dataset with Wikipedia descriptions.

4.2 Wikipedia

Since Wikipedia could give some official explanations of items, it is a great attempt to collect items' corresponded elaboration in there. The item names we get from Wiki has some overlap with the ones in Amazon Dataset in order to help the fine-tuning process. We get the data in a similar way as we do for Amazon Dataset. Due to the fact that the data is clean enough, it would be very convenient to grab the first paragraph for each term page as the description without further format adjustment. The final

[1] https://jmcauley.ucsd.edu/data/amazon/

fine-tuning data for base generator is composed of 50 wikis and 50 Amazon data.

4.3 LIGHT

We briefly discuss the LIGHT dataset[2] since we frequently work with this dataset throughout this course. At this stage, we only use the objects dataset. For each one of the game items, there are `'name'`, `'in_room_ids'`, `'base_form'`, and `'descriptions'`. To formulate a style transfer task, we first apply the pre-constructed base generator to `'base_form'` to make a pseudo-parallel dataset. Next, for the rooms an item exists in, we use the `'category'` of the rooms to control the style transfer (use as `'setting'`). Finally, we have the required information for the renderer.

4.4 Dark Souls

Dark Souls' Wiki is known for its speciality in illustrating item. For example, it always formally describes one item at first and then show some related but a little hilarious stuff. Hence, this data is specially prepared for those games which need that style. We crawl those data from the Dark Souls Wiki website[3] by taking use of the property of HTML. Weapons are our focused crawling target. Totally, there are over 200 weapon info scraped from Dark Souls 3.

5 Evaluation

Metrics The most important component of our system is the renderer. We primarily focus on evaluating lexical overlapping and semantic preservation. We use the **BLEU Score** (Papineni et al., 2002) to measure the lexical overlap. To evaluate the degree of semantic preservation, we plan to use **BERTScore** (Zhang* et al., 2020) and **Sentence-BERT** (Reimers and Gurevych, 2019) to compute the semantic similarity between the generated text and the ground truth. The perfect result would be the two pieces of texts being almost paraphrases of each other. We discard **Perplexity** evaluation because the game item descriptions may naturally be very complex because the "uncommon words," i.e., game-specific words, are very common.

Besides automated evaluations, we consider human evaluations to determine the quality of the generated item descriptions. We proposed 3 criterion for human evaluation: overall quality, consistency with item name, and consistency with game category. Each criteria uses 5 point rating scale, where a score of 1 means the worst, and a score of 5 means the best. We plan to use anonymous Google forms to let people rate our generated descriptions. At current time, we have not yet passed the forms out to people and we plan to do it in this week.

Test Set Sampling To evaluate on the LIGHT objects dataset, we randomly select 10% base forms such as *trees, cookpot,* and *stones*. Then, we use the fine-tuned GPT-3 to generate the base descriptions for these selected base items. Next, for all items belonging to these sampled base forms, we apply the renderer to produce game object descriptions and evaluate the generation quality using the above metrics. There are 99 base forms and 209 LIGHT game items in total.

Evaluation Parameters For BERTScore, we use `roberta-large` as the encoder, which is the default setting. For paraphrase detection using SentenceBERT, we use `paraphrase-albert-small-v2` for sentence encoding. Then, given the two embedding for LIGHT object description and generated description, we compute the cosine similarity between them. Lastly, for the BLEU Score, we first tokenize the descriptions with `wordpunct_tokenize`, then lemmatize the tokens with `WordNetLemmatizer` and remove stop words, using `nltk` (Bird et al., 2009). The bi-gram and tri-gram scores are almost exclusively zero. Thus, we only report uni-gram BLEU score.

6 Results

6.1 General Renderer Results

For our renderer, after fine-tuning, we observed some promising results. Using our renderer, we are able to generate full descriptions that are different in style, when the game category changes. We put a few examples in Table 1 below. From these examples, we can clearly see that in the **Abandoned** game setting, the description of the **Excalibur sword** includes phrases like "old and worn". This description is very differently from the description in the **Dungeon** game setting. In addition, in the **Wasteland** game setting, the description of **haunted totem** includes phrases like "is old and creaked when they were touched". And when the game setting is changed to **netherworld**, the description includes phrases like "carved from the heart of a woman", which fits the category of

[2]https://parl.ai/projects/light/
[3]https://darksouls3.wiki.fextralife.com

netherworld. Such evidence suggests that our renderer shows some degree of text style transfer.

6.2 Dark Souls Renderer Results

For this special renderer, the fine-tuning model works as expected. Given an item, we can generate dark-souls-like style descriptions. We put a few examples in Table 2 below. For **king arthur sword**, the result includes the basic info about this sword as well as some deeper-level explanation about that weapon like its impact and some story telling. Also, for some items already existed in Dark Souls, like **Standard Helm**, it shows not only its basic property but also some functionality out of surprise.

6.3 Evaluation Results

The automatic evaluation results based on five mentioned automated evaluation metrics are shown in Table 3.

7 Discussion

This project revisits the game-item-description generation problem using large pre-trained language models. We decompose the end-to-end generation task into base description generation and text style rendering. The overall generation quality is satisfactory. We attribute this result to the "No Free Lunch" theorem. Using two separate models for generation and rendering allows the models to encode specialized information.

Meanwhile, we also identify several problems, which shed light on potential future research directions. Firstly, the base generator is not stable enough. We notice that, when re-generate the descriptions several times, the model may produce completely different texts. Further, the rendering consistency still needs improvement. Although the rendered texts can take up some fantasy game flavor, they are not based on a consistent game setting. For instance, we know that in many games, the game items together help construct the core information of the game world. The objects' descriptions reveal the background stories of related entities like gods or kings. So, we hypothesize that an entity linking module and external knowledge base (explicitly encoding the world setting) can be beneficial in our future work.

Acknowledgements

We would like to thank Professor Chris Callison-Burch, Dr. Lara J. Martin, Teaching Assistant Liam Dugan and Teaching Assistant Artemis Panagopoulou for their support and guidance. We would especially like to express thanks to TA Liam Dugan for his guidance and insight in our project.

Item	Category	Description
Excalibur sword	Abandoned	The sword is old and worn from use. The hilt is adorned with a seal that clearly shows the name of the late king.
Excalibur sword	Dungeon	The Excalibur sword is the kings best weapon. It is sharp as a tack, and its black blade is as sharp as any person's eye.
haunted totem	Wasteland	The totem is carved from a dark wood and has a twisted black design. The wood is old and creaked when they were touched.
haunted totem	netherworld	The totem has been carved from the heart of a woman who has seen and done it all. The stone around the stone circle is carved from the blood of her enemies and the wood is made of choice wood from the forest.

Table 1: Examples of generated descriptions of the same item in different game settings.

Item	Description
king arthur sword	A powerful and well-crafted sword that requires great strength to wield. The king's sword, and the only sword that was permitted to be wielded by the king, required the most strength to wield. Skill: Stance. While in stance, use normal attack to break a foe's guard from below, and strong attack to slash upwards with a forward lunge.
Standard Helm	Standard helmet with a large, armored visor. Must be equipped in order to useanced skill. Skill: War-cry. Let out a spirited warcry that temporarily boost poise.

Table 2: Examples of generated descriptions of items in Dark-Souls-like style.

Metric	Mean	Stdev.
BLEU	0.191	0.123
Paraphrase	0.524	0.156
BERT-Precision	0.879	0.028
BERT-Recall	0.886	0.035
BERT-F1	0.882	0.026

Table 3: Renderer quality automated evaluation over LIGHT dataset.

References

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

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. 2020. Language models are few-shot learners. *CoRR*, abs/2005.14165.

Suchin Gururangan, Ana Marasovic, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. *CoRR*, abs/2004.10964.

Jena D. Hwang, Chandra Bhagavatula, Ronan Le Bras, Jeff Da, Keisuke Sakaguchi, Antoine Bosselut, and Yejin Choi. 2020. COMET-ATOMIC 2020: On symbolic and neural commonsense knowledge graphs. *CoRR*, abs/2010.05953.

Di Jin, Zhijing Jin, Zhiting Hu, Olga Vechtomova, and Rada Mihalcea. 2020. Deep learning for text style transfer: A survey. *CoRR*, abs/2011.00416.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Fabio Petroni, Tim Rocktäschel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. 2019. Language models as knowledge bases? *CoRR*, abs/1909.01066.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

Sudha Rao and Joel R. Tetreault. 2018. Dear sir or madam, may I introduce the YAFC corpus: Corpus, benchmarks and metrics for formality style transfer. *CoRR*, abs/1803.06535.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*.