ECE1786H Creative Application of NLP
Final Report

"*SoulsGen*" - Generating Items in Dark Souls styles

Haocheng Wei (1008498261)
Kaifeng Meng (1007923945)

Total Word Count: 1749
Compute Penalty: 0%

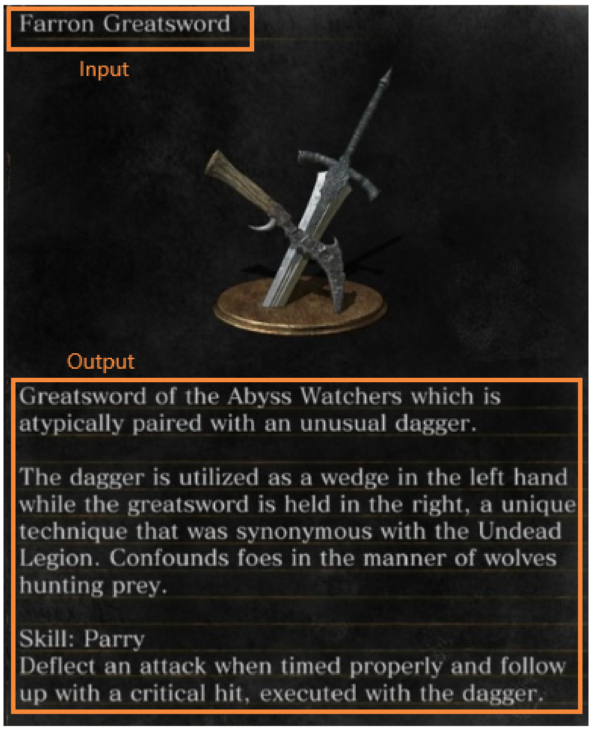
1. Introduction

Our project focuses on the stylized text generator model – *SoulsGen*. It is a transformer that takes a prompt (i.e. an item or an object's name) as input while generating lines of descriptions of it, flavored in the writing style of the game series Dark Souls.[1]

The style we are learning is how they write description text of an item (weapons, sorceries, etc.) in the game, which often includes 3 parts: what it is, how to use it, and the background that provides scattered details of the western fantasy-styled plot of the game.

The model is trained on the DSItems dataset built from scratch by ourselves in this project, and is able to leverage latent information in the input together with the user suggestion to generate the game-flavored descriptive context.

The expected application of the model is mainly for recreational usage for the game fans community, while it can also provide conceptual ideas for writers to describe their own items when writing a western-fantasy styled story.



Farron Greatsword

Greatsword of the Abyss Watchers which is atypically paired with an unusual dagger.

The dagger is utilized as a wedge in the left hand while the greatsword is held in the right, a unique technique that was synonymous with the Undead Legion. Confounds foes in the manner of wolves hunting prey.

Skill: Parry
Deflect an attack when timed properly and follow up with a critical hit, executed with the dagger.

Fig 1. An example of the input (title) and stylized output (contents)

2. Background

Text Generation is a very challenging NLP topic. The tasks of text generation vary from automatic fill-ins to automatic chatbots. When performing sequential generation tasks, Recurrent Neural Network (RNN) [1] is the most widely-used model. Besides, MaskGAN [2] provides a GAN based approach. The famous GPT model [3] is currently the largest and

[1] For more about the game-flavored text and dialogue, see: https://en.wikipedia.org/wiki/Dark_Souls

most well-trained text generation model which can perform almost any text generating task. In this project, we use the TextGenRNN [4] model as our baseline, and implement our SoulsGen model based on GPT-2.

While our model focuses on stylized text generation rather than plain text generation, text style transfer for artistic writing and communication is another similar topic [5]. A major approach is to train on parallel data to learn the conversion between different writing styles. The definition and classification of “styles” is surveyed by Jin, et al. [6]

3. Data Processing

3.1 Source of Data

We collect the dataset from scratch. The text data comes from a non-official game wiki, where each entry has a separate wiki page. Therefore, the raw data is collected by downloading the source code of the wiki pages.

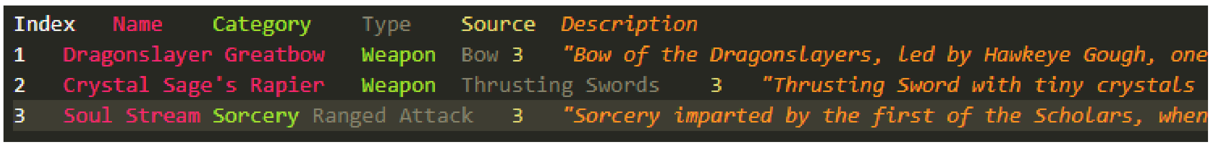
3.2 Data Processing

The extracted text data includes the name, attributes, source (from which game), and description of an item. All of the text data is wrapped in an HTML container, each column shares the same container exclusive from other columns.

We created a preprocessor to extract text data from the source code using regular expressions. Then save the data using .tsv format or further process. The dataset includes 2200+ sample items, from a majorly 6 different categories (weapons, sorceries, consumables, etc.) in 4 games of the series. The dataset is further splitted into train and validation dataset randomly, with the training set and validation set at 80% and 20% percent of the rest respectively. The test set is independent from the dataset, using 80 samples of the same attributes from another game in the series.

When creating the dataset, we hope that the category, type, and source features we collected can help us capture the minor difference between different item types or game series, but not all the features will be used eventually.

We clean the data by filtering out the empty or `<null>` descriptions, and we also discard unwanted words: HTML tags and entities (e.g. `<em>` and `&nbsp;`). As the data is from a series rather than a single game, the duplicate of item names from the different games is observed and discarded based on the time priority. We use the BPETokenizer to convert the words into indices.



```
Index   Name    Category    Type    Source  Description
1   Dragonslayer Greatbow   Weapon  Bow 3   "Bow of the Dragonslayers, led by Hawkeye Gough, one
2   Crystal Sage's Rapier   Weapon  Thrusting Swords    3   "Thrusting Sword with tiny crystals
3   Soul Stream Sorcery Ranged Attack   3   "Sorcery imparted by the first of the Scholars, when
```

Figure 2. Example of the extracted and cleaned dataset

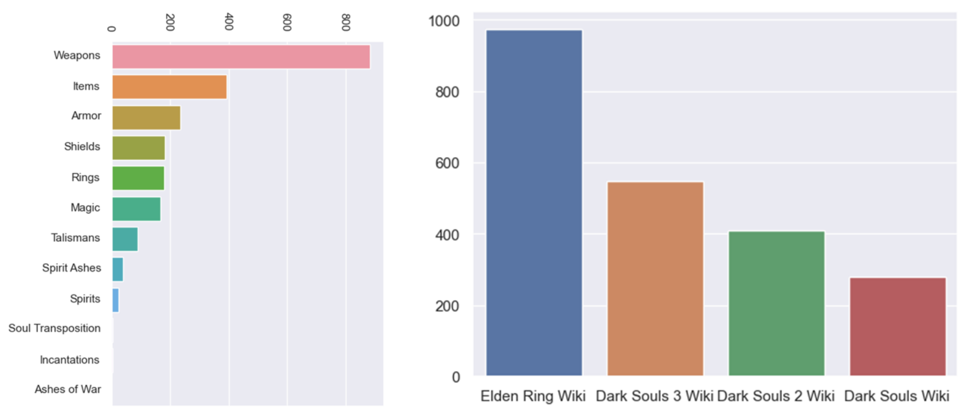


Figure 3. Data visualization
Left: Category of sample data.
Right: Original game (site) of sample data.

4. Baseline Model

Among many word-level NLG models for different generation tasks, we choose TextGenRNN model as our baseline model. The TextGenRNN model is originally used to generate fake Reddit articles given the title of the article, which is similar to our usage to generate item description given the item name. In fact, we do not even need to change the architecture of the model or adjust our data format when training the model.

The model contains 2 LSTM blocks and concatenates the residuals of each as well as the embedding layer. Figure 3 shows the structure of the TextGenRNN model. The model structure remained unchanged while we fine-tuned the hyper-parameters. The parameters of the best model on the test set is shown in table 1.

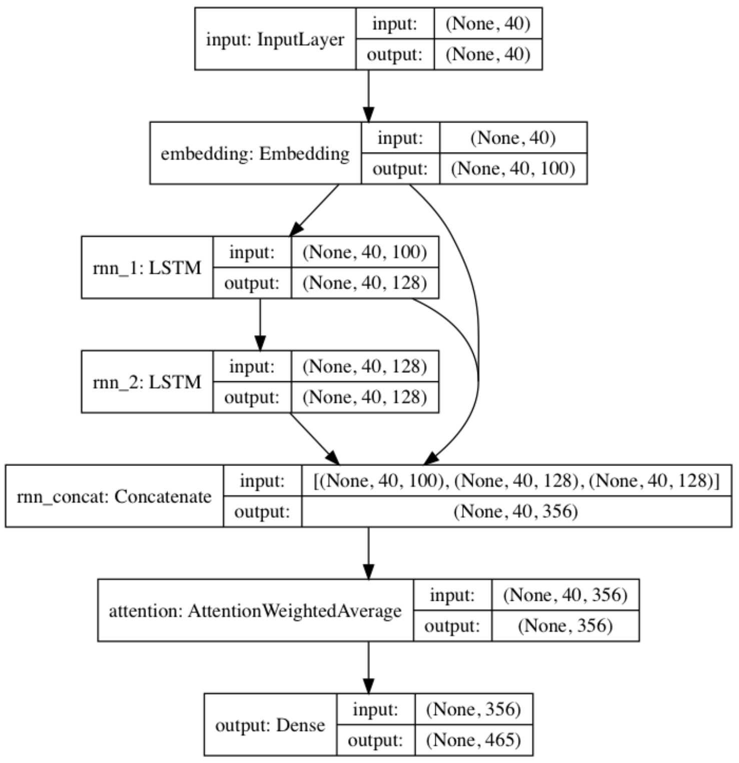

input: InputLayer | input: (None, 40) | output: (None, 40)
embedding: Embedding | input: (None, 40) | output: (None, 40, 100)
rnn_1: LSTM | input: (None, 40, 100) | output: (None, 40, 128)
rnn_2: LSTM | input: (None, 40, 128) | output: (None, 40, 128)
rnn_concat: Concatenate | input: [(None, 40, 100), (None, 40, 128), (None, 40, 128)] | output: (None, 40, 356)
attention: AttentionWeightedAverage | input: (None, 40, 356) | output: (None, 356)
output: Dense | input: (None, 356) | output: (None, 465)

Figure 3. Structure of the Baseline model

Table 1. Parameters used in Baseline model

Optimizer	Learning Rate	Batch Size	Max Length	Epochs
Adam	4e-3	64	300	30

The result on the test set is shown in section 5. When generating output using the fine-tuned model, the generation speed is very slow, while the result is not flawless. Therefore, we decided to build our own GPT-2 based generator for this task.

5. SoulsGen Architecture

Our generative model is fine-tuned from HuggingFace's standard GPT-2 (124M parameters version) under the GPTLMHeadModel and also with the corresponding tokenizer, which is designed for text generation tasks. An overview of a typical GPT-2 model structure is shown below (figure 4), and the hyperparameters used in our fine-tuning (table 2).

Table 2. Hyperparameters used for training GPT-2

Learning Rate	Batch Size	Max Length	Epochs
2e-4	2	100	3

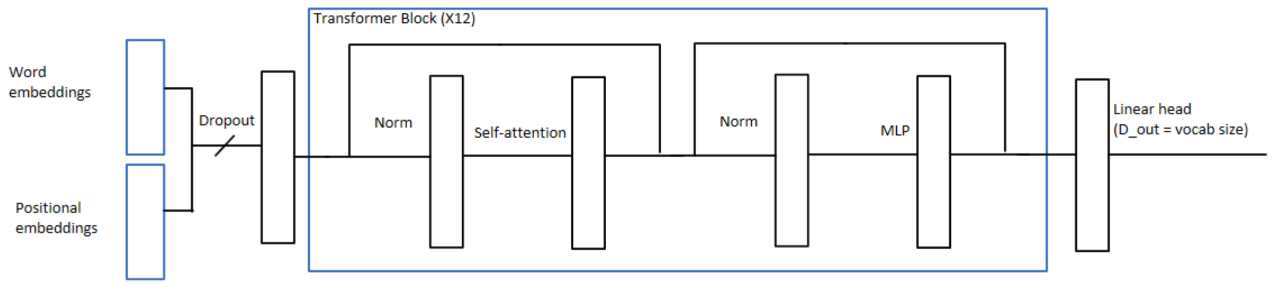


Figure 4. Structure of the Transformer model (GPT-2)

6. Result and Evaluation

6.1 Model Based Metric

To measure the performance of our generative model quantitatively, we have trained a separate GPT-2 based classifier that can give us a scalar result indicating the confidence of whether the generated context has an expected style.

The classifier has been trained on a dataset consisting of descriptive plain English as negative samples and an equivalent number of examples randomly picked from Dark Souls dataset as positive samples. The plain English samples are generated from GPT-3 davinci-003 model by asking it to describe the game items with their original categories provided ("Describe XXX as weapon" etc.), as we believe the powerful GPT-3 model is capable of generating good enough descriptive context as humans can do.

Due to the various category sizes as well as their slight difference context formats, we expect to observe some bias on our classifier when inputs are from, or suggested to be in different categories. After the training, we evaluate our classifier on in-game data (Dark Souls dataset), out-game data (Demon Souls test dataset, not included in any training), and Dark Souls' category specific datasets. The results are given in the following table.

Table 3. Classifier evaluation results

Dataset	Confidence
Training set average	0.8439
Test set average	0.9745
weapon category	0.8419
shield category	0.7419
armor category	0.9198
item category	0.7433
ring category	0.9861
misc category	0.8869

Therefore, about 0.8 score is expected to confidently say the generated context has a Dark Souls style, lower scores may be tolerated for specific categories.

6.2 Quantitative Results

A test set has been constructed to evaluate and compare the performance of SoulsGen and the baseline model. The test set contains 50 handwritten samples considered as general objects ("A Cup of Water", "Green Hat" etc.), entire Demon Souls dataset as non-existed game objects and 100 items randomly selected from Dark Souls dataset as typical game objects. The results are shown in Table 4 below.

Table 4. SoulsGen and Baseline evaluation results

Dataset	SoulsGen	Baseline
Overall Average	0.93	0.76
Test Dataset	0.91	0.80
Handwritten	0.99	0.60
Main Dataset	0.89	0.89

Another observation we found during evaluation was the time cost for single generation. Under the default environment of Google Colab, when generating up to 100 tokens with temperature 0.5 ~ 0.7, SoulsGen spent approximately only 1 second to complete the generation while the baseline model was about 9 seconds.

These results show that our SoulsGen model, compared to the baseline, can have a better performance in generating Dark Souls style context especially from general objects, and also much faster.

6.3 Qualitative Results

Our input prompt is expected to be an object ("A Cup of Water" etc.) or an object's name ("Adam's Cup" etc.), with an optional suggestion tag ("A Cup of Water [Magic]" etc.), which is supposed to be one of the valid categories. The generator will describe the object as suggested if the tag is provided, otherwise it will generate a tag as part of its generation context. Some sample outputs from our generator are shown in table 5.

Table 5. Sample outputs (as expected)

Input	Output	Score
A Cup of **Water**	*[**Item**]* *A cup of warm **water**. Craftable item.* *Temporarily boosts resistance to poison.* *Thought to be a **smoldering pot**.*	0.9444
A Cup of **Water** [**Magic**]	*One of the glintstone **sorceries** of the **Academy** of Raya Lucaria.* *Creates **a large, floating fountain of water**.* ***This sorcery can be cast while in motion**.* *The glintstone sorceries of the academy are an extension of the Elden Ring, the ancient art of co-operating.*	0.9715
Dragonslayer Great**bow**	*[**Weapons**]* *Great**bow** fashioned from the bones of the Dragonslayer, who once served Lord Gwyn.* *The ancient dragon cultists believed that this greatbow, fashioned from a dragon, served to **ward off** the Darkwraiths and other vengeful spirits, and would restore the Undead Legion.* ***Skill**: Dragon Breath* ***Burrow** into foes with a powerful **attack** to greatly restore HP.*	0.9459
Dragonslayer Great**bow** [**Magic**]	*A great**bow imbued** with the **power of** the Godskin. Throw at enemies to cause buildup of **fire**.* ***This sorcery can be cast while in motion.*** *The ancient Dragonsmen wielded great **arrows**, and these were wielded by their Lords.*	0.9094

Above examples are when the model performs as our expected:

- It will follow the suggestion tag if provided
- It will leverage the information from the input prompt to generate reasonable tag and somehow related context

The first expectation, as marked in **blue**, shows the special words or formats that usually along with the tags, such as "weapons" are used for attack and would have a skill, or "magic" need to be "cast" or related to some special power. Those terminologies or formats of a specific tag are what we expect to observe in the context.

As for the second, marked in **green**, shows that our generator can not only use the prompt as an object name, but also could generate a highly related context or tag. Such as the"bow" should be considered as "weapons", and "bow" would use "arrows" to "burrow". And if the tag does not match up with the common sense of the object, it will result in a mixture of the style, such as a bow imbued with a magic power.

However, as mentioned in the previous section, there are some categories that have relatively less data than others, which might result in high repetition rate of certain words, or formats to be observed in generations, shown in table 6.

Table 6. Sample outputs (can be improved)

Input	Output	Score
Adam's Guard [**Spirits**]	***Ashen remains in which spirits yet dwell**. Use to summon the spirits of four Fingers.* *Fingers must be equipped to use the **Spirit of Fingers**.* ***Spirit of Fingers*** *Fingers who peer into the depths of chaos. The souls that inhabit them cling to the threshold of their own being.*	0.9153
A Cup of Sand [**Spirits**]	***Ashen remains in which spirits yet dwell**.Use to summon a frost giant.* *In the Undead Settlement, the Giants lurk in the shadows, guarding the ruined cathedral.*	0.8972

A Cup of Sand [**Magic**]	*One of the incantations of the Academy of Raya Lucaria.* *Summons a cloud of sand, then fires a stream of projectiles.* *Charging extends duration.* ***This incantation can be cast while in motion.*** *The academy is a land of ancient sorcerers, and the sand is their weapon of choice.*	0.9990
Dragonslayer Great**bow** [**Shields**]	*Greatbow wielded by the Dragonslayer Apostles.Greatbow of unknown origin, originating from a dragon's blood.* *The Apostles were known as the Dearborn and their training was widely studied, until the Dragon Communion, when they were scattered across the Lands Between.*	0.88

The first two examples show a typical **repetition** that can be observed almost in all “[Spirits]” generations, and the “This incantation can be cast while in motion” in “[Magic]” generations is another example. Although they are truly Dark Souls styled, from the perspective of gamers, this is not what we want to see in every description.

And another case when the model cannot work as expected is when the common sense of the object is too far away from the given tag, the example in the last row shows a latent **conflict** between object name and suggestion, resulting in the generated context having no relation to the “[Shields]”.

7. Discussion and Learning

7.1 Discussion on results

According to the results analyzed, most of the results on the SoulsGen model meet our expectation for the “dark souls style” description. While the baseline model also provides many trustable results, the quantitative score of the baseline model is surprisingly under our expectation. Also, we didn’t expect that our model, which has far more weights than the baseline, runs faster when generating outputs.

7.2 Future works

In future work, we will mainly focus on improving the model by conducting manual labels on the outputs. By putting the model online and implementing a rating system for the output and collecting the rating data we can do such manual labeling work at low cost. Also, improvements are needed for the classification model, since “being far away from plain text” is not all that we want for the output.

8. Individual Contribution

The individual contribution is shown in table 7

Table 7. Individual contribution

Haocheng Wei	Kaifeng Meng
• Responsible for conducting data collection and preprocessing.[2] • Responsible for early research. • Responsible for baseline model implementation and tuning.	• Responsible for the transformer model implementation and tuning. • Responsible for research on performance of different GPT models.

[2] The dataset is open-sourced, and has already had some progress before the project begins. The processor along with the processed dataset is later uploaded by Kaifeng.

● Responsible for research on different evaluation metrics. ● Implemented non-model-based metrics (used in midterm report) ● Responsible for project and version management.	● Responsible for the classification model implementation. ● Mainly responsible for result analysis. ● Help collected 30% of the raw data from the game wiki. ● Help improve the logic when performing data cleaning. ● Others

9. References

[1] K. Gregor, I. Danihelka, A. Graves, D. Rezende and D. Wierstra, "DRAW: A Recurrent Neural Network For Image Generation," *32nd International Conference on Machine Learning,* pp. 1462-1471, 2015.

[2] W. Fedus, I. Goodfellow and A. M. Dai, "MaskGAN: Better Text Generation via Filling in the______," 1 Mar 2018. [Online]. Available: https://arxiv.org/abs/1801.07736.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[4] minimaxir, "Textgenrnn: Easily Train Your Own Text-generating Neural Network of Any Size and Complexity on Any Text Dataset With a Few Lines of Code.," GitHub, [Online]. Available: github.com/minimaxir/textgenrnn.

[5] A. Celikyilmaz, E. Clark and J. Gao, "Evaluation of Text Generation: A Survey," 18 May 2021. [Online]. Available: https://arxiv.org/abs/2006.14799.

[6] D. Jin, Z. Jin, Z. Hu, O. Vechtomova and R. Mihalcea, "Deep Learning for Text Style Transfer: A Survey," *Computational Linguistics,* p. 155–205, 2022.

Permissions

Permission from ***Haocheng Wei*** to post:

Video:	yes
Final report:	yes
Source code:	yes

Permission from ***Kaifeng Meng*** to post:

Video:	wait
Final report:	yes
Source code:	yes