Is GPT All We Need for Situation Puzzle System?

Xinyue Wang*[1], Xuanyu Wu*[1], Yongzhe Zhu*[2], Zhaoyi Hou*[1]

University of Pennsylvania
Philadelphia, Pennsylvania 19104
[1] {wsinyue, xuanyuwu, joeyhou}@seas.upenn.edu, [2] zhuyz@sas.upenn.edu

1. Introduction

Situation puzzle is a popular text-based interactive game. In such a game, there is a moderator and a player, or a group of players. The moderator has a complete story in mind and provides the player(s) with the beginning and the ending of a story; the player(s), on the other hand, are expected to figure out the step-by-step details of story development by asking the moderator ”yes-or-no” questions about the story. The game would end either when the player(s) figure out the complete story line or when a preset number of questions are asked before complete the story. The challenging and intriguing part of this game is that the player(s) need to come up with a story with only the beginning and the ending while also make sure the logical relations in the story align with commonsense.

We propose to build a system (TurtleSoup) that not only automatically generate high-quality story for a situation puzzle game but also handles the job of the moderator to interact with player(s). With minimum human intervention, TurtleSoup is capable of providing comprehensive interactive experience that can be entertaining.

2. Related work

The TurtleSoup can be divided into two main components: story generation and question-answering for game state tracking, both of which are well-studies topics in natural language processing in recent years.

2.1 Prompt Engineering for Story Generation

In the past few years, especially with the advent of large language models, such as GPT-3 [Brown et al.2020], prompt engineering [Liu et al.2021] become widely adopted in various generic language tasks, such as dialogues, question answering, and text generation. Prior to input into the language model, a prompt template was pre-filled with the core information to better guide the model on the generation. By working on different prompts, the same language model can perform multiple tasks with fine-grained controllability on the style of the results. By providing more information in the form of prompt structure and content, language models can gain more information for its task and in turn help subtle problem solving which required fine-tuning the data beforehand [Trinh and Le2019]. In our project, we will continue this way to carefully construct sequence of prompts, in order to generate our story in a guided and controlled manner.

2.2 Storyline-Based Generation

There are numerous approaches to generate a story. For our particular task, logically coherence story is the key. One way of ensure logical coherence is to generate a storyline before turning it into natural language. The GraphPlan [Chen et al.2021] and Plan-and-Write [Yao et al.2019] both explore this paradigm and achieved decent results.

In the GraphPlan [Chen et al.2021], a Seq2Seq model is built to measure the level of causal coherence between any two events. Based on the coherence score between the previous event and all the candidate events, the event with highest score would be chosen to be the next event, until a preset number events is reached in this storyline. One of the drawback, though, with GraphPlan, is that the candidate events are built from previously exacted stories, which means the variety of the generated events heavily depend of this event pool. In Plan-and-Write [Yao et al.2019], a storyline is also built to aid logical story generation. The main difference is that the events on the storyline is also generated via neural language model (BiLSTM) given previous event phrase and the title. This key difference makes it capable of generating out-of-scope events while also increases the complexity and runtime of the overall system.

2.3 Genre Representation

Another related topi is the representation of genre, or topics, of stories. That is because one of the key features of TurtleSoup is to generate surprising stories and therefore amaze the users. To accomplish that, a precise embedding of particular genre is required. In this area, both traditional methods like latent Dirichlet allocation, or LDA [Jelodar et al.2019], and neural-net-based models such as the TopNet [Yang et al.2021] could achieve great results. In particular, the TopNet provides dynamic embedding for topics and flexibility for integration to the generation model.

*These authors contributed equally.

model	text-davinci-002
temperature	0.7
max_tokens Layers	256
top_p	0.7
frequency_penalty	0.5
presence_penalty	0.6

Table 1: GPT-3 Generation Settings

2.4 Question Answering

Question answering given context has been a well-studied field in natural language understanding. In our setting, the goal of question answering is to test the entailment relationship between player's query and events in the generated story, which is hidden from the player. BoolQ [Clark et al.2019] explores this particular problem and provided a reading comprehension dataset for yes-and-no question. In terms of software, HuggingFace provides BertForSequenceClassification [hug] that can handle this task of yes-and-no classification given context (the story) and question (from player).

3. Coherence Story Generation

The first part of TurtleSoup is the story generation module, which generate a series of natural language sentences given a topic phrase. Put it formally: given a starting sentence s_0, generate a preset number (n) of sentences to complete the story $(s_1, s_2, ...s_n)$.

The key part of coherence story generation is to make sure that any two consecutive sentences are coherent to each other. In other words, the previous event should inform the next one in a logical manner. Based on that requirement, we tried the following methods for story generation.

3.1 Baseline

As our baseline method for story generation, we used GPT-3 for zero-shot story generation. We give the prompt in the following format:

> "[Beginning of the Story] What happened next? Write a surprising story."

See Table 1 for the settings of GPT-3 generation. Note that, unless otherwise specified, all the story completion generation in the following sections all adopt the same set of parameters.

3.2 Controlled Story Generation with GPT-3

The second method we implemented is also mainly based on GPT-3 model for story generation. The overall idea is to generate story with several steps (i.e. multiple calls to the GPT-3 model) so that we can enforce controls for each generation. To put it formally, given the beginning sentence s_0, the $i-th$ call to the GPT-3 model would have a prompt that includes all the existing sentences so far $(s_0, s_1, s_2, ...s_{i-1})$.

Besides the step-by-step style of story generation, what makes this model unique is the prompt that we engineered for each call to the GPT-3 model. All of the prompts in those multiple calls follow the following template:

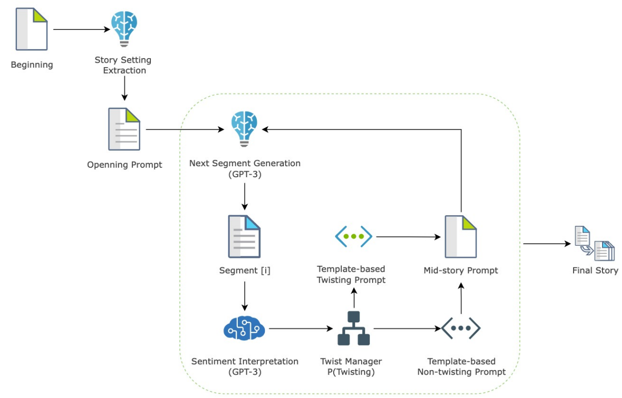

Figure 1: Controlled Story Generation with GPT-3

> {Story Preset}
> {Story content so far $(s_0, s_1, s_2, ...s_{i-1})$}
> {Transitional Prompt}

The "{Story Preset}" and the "{Transitional Prompt}" will be discussed in Section 3.2.1 and Section 3.2.3, respectively.

Other than prompt engineering, we also introduced "Sentiment Interpretation" module and "New Sentence Verification" module, which will be discussed in Section 3.2.2 and Section 3.2.4, respectively. See Figure 1 for the end-to-end procedure.

3.2.1 Story Preset Engineering The main goal of story preset engineering is to provide GPT-3 model additional information (preset information) in the prompt such that it would have a better chance of generating content-rich story. The preset information is a series of text based on the following template:

> Setting:
> Location: {location}
> Main Character: {main character(s)}
> Other Characters: {possible character(s)}
> Genre: {genre}
> {Story content so far}

Both the very first prompt and all the following ones would include this set of text.

In this preset, the first three pieces of information are gathered through a zero-shot GPT-3 completion task. For example, for the location information, we require GPT-3 to complete the following prompt:

> What is the most likely location for the following story: (Our First sentence)

Then, we strip out unnecessary information (e.g. "The most likely location for the story is"), and save the remaining text as the story preset information.

The last part of the story preset, genre, is randomly selected from the following pre-defined genres: suspense, detective, comedy, supernatural, fantasy, adventure, city legend, puzzle.

3.2.2 Sentiment Interpretation A great story reversal is highly related to its sentiment change, which is stronger when the huge sentiment change happens in a reasonable way. To simulate this structure, we introduce the sentiment module to classify the sentiment of each sentence and quantify it for better reversal sentiment generation. Here we utilized two sentiment exactor, GPT-3 and sentence transformer pretrained on twitter three-class sentiment classification task. By setting ten kinds of concrete sentiments such as happy and surprising, and asking GPT-3 which sentiment could be inferred from current sentence, the sentiment category could be specified. Since these ten classes of sentiments could be induced to more general three classes of sentiments(positive, neutral, negative), the classification probability outputted by sentence transformer can appropriately represent the strength of a particular sentiment. In order to better convert quantity into prompts, we discretized the probability into ten levels including ”not very”, ”slightly”, ”very”, ”extremely” and etc, and these adverbs add-ins are proved to be effective on few-shot experiments. Combining the corresponding adverb and the reversal of the sentiment of current sentiment, we can get a more powerful prompt to generate a more dramastic and twisting following story.

Note that one of the advantages here we adopt prompt engineering and large generative model like GPT-3 is that even with strong sentiment reversal prompt inserted, these reversal contents are generated in a reasonable and internally consistent way and it is important in situation puzzle task, because a good situation puzzle should be surprising and logically coherent simultaneously.

3.2.3 Story Twist Generation Beyond the sentiment interpretation, it is vital to mitigate the repetition of constant reversals and improve the coherence of story, which prevents the surprising fatigue of players. We control the twisting generation by utilizing two parameters, twisting probability and twisting step. While the former one is used to decide whether this sentence will be exacted sentiment, the latter one is a factor deciding when to execute twisting probability sampling between how many segments generation. These hyperparameters are tuned based on human evaluation of generated story by ablation experiments, which indicates that the twisting is most interesting and coherent when twisting probability is 0.7 and twisting step is 1. Here we adopt the following template as our twisting prompts templates and insert the sentiment analysis results to generate the concrete twisting prompts.

Then, something [Adverb] [Sentiment] happened.

To fully utilize the surprisingness GPT-3 itself has, we set the ”top_p” and ”temperature” to be 0.7, while not using stop words to get more complete generation and mitigate the internal repetitive mechanism GPT-3 has.

In addition to twisting prompts, adopting continuous prompt when no twisting is central to generate a good situation puzzle as well. Note that language generation model sometimes inevitably generates highly similar, even repetitive contents and mistakenly regards the still going-on story as already ended, it is essential to remind it that the story is still moving forward and the contents needs to be effectively updated. To achieve this goal, we introduce continuous prompts such as ”Then,”, ”After a while,”, ”Meanwhile,”, ”And then,” and etc, which is shown effectively decreasing the repeated contents and hinder the empty generation happened.

3.2.4 New Content Verification Each time we call the GPT-3 model, a completion to the current story is generated. However, new content verification, which is a two-step process, is required before the generated completion is added to the story. For first step, we would like to ensure the final story would not be too long. The way to accomplish that is simply to truncate the newly generated content to no more than three sentences.

After that, we implemented a very basic duplicate check via sentence similarity. The motivation is to ensure the newly generated content is not a simple repetition or paraphrase to the previous content. The way we measure content similarity is the following: first, embed each sentence with SentenceBert [Reimers and Gurevych2019] for each sentence in the newly generated segment (no more than three sentences) and also each sentence in the existing story; then, cosine similarity is calculated between each embedding in the newly generated segment and each existing story where the top similarity among those sentence pair is selected to represent similarity between generated segment and the existing story. If any of those cosine similarity scores between one of the newly generated sentences and one of the existing sentences is higher than 0.9, we will call the GPT-3 model again, with a different, randomly selected transitional prompt; if more than 5 such re-generations are executed and the issue persists, we will pick the generation results with lowest top similarity score and append that result to the current story. Note that, empirical result shows that our model rarely go into this deadlock.

3.3 Event-to-event Story Generation with LSTM

The second method we implement for story generation is a three-step approach First, extract the main event from the first sentence into a tuple $(Subject, Verb, Object, Modifier)$; then, given the extracted first event, generate a series of events which themselves are coherent to each other; given the generated events, generate one natural language sentence per event, which would connecting them into a complete story. See Figure 2 for more details.

3.3.1 Event Extraction The event extraction module transforms a given sentence into a tuple with four tokens (in string): $< Subject, Verb, Object, Modifier >$. In particular, the words in the tuple are all lemmatized.
Aside from the original version of the event extraction, we also added a second version of event extraction, as per [Martin et al.2018] did: we further generalized the words in this representation by the following steps:

- Replace the subject with recognized name entities (e.g. $< PERSON >$, $< NE > n$, $< PRP >$) [Finkel, Grenager, and Manning2005];

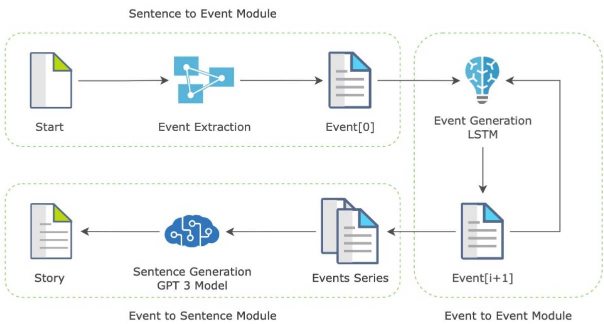

Figure 2: Event-to-event Story Generation with LSTM

Parameter	Original	Generalized
Token Emb. Dim.	512	512
LSTM Layers	4	4
Hidden Layer Dim.	1024	1024
Token Vocab Size	28029	10372

Table 2: Event-to-event LSTM Model Setup

- The rest of the nouns are replaced by the WordNet [Miller1995] synset two levels up;
- Replace the verb with VerbNet frames from [Schuler2005]

3.3.2 Next Event Generation The second step, generate an event given previous event, is done by a event-to-event LSTM model trained on a dataset of science fiction TV show plot synopses, scraped from Fandom.com wikis [Ammanabrolu et al.2020]. Since the first step, event extraction, produces two different types of outcomes: original-text events and generalized-text events, two event-to-event LSTM models are trained separately, with corresponding data. The model setup are exactly the same for two models, except for the vocabulary size (see Table 2).

With the trained LSTM and the extracted events from the first sentence, the event-to-event LSTM would generate a series of events until a given number of events is generated. That would conclude the event generation part.

3.3.3 Event to Natural Language Generation In any of the methods above, our final outcome from the network would be separated tuples of events. The event-to-story generation requires simply writing a sentence that extends and combines the (single or multiple) 5-tuples into a full sentence. (In prospect, this generation might also refer to the global states - places, characters, actions - we might additionally keep in the generation process.)

We first tried to use GPT-2 with fewer parameters to see if we can maximize performance with less cost. However, the GPT-2 generation were not satisfactory, as the model does not semantically understand the prompt, and only manages to parse the tuple as separate words, attempting to write a full story starting with these words.

Thus we still turn to GPT-3 as the Event2Sentence generation method. We give the prompt in the following format:

> Generate a sentence using the following words:
> [cleaned 5-tuple]
> ###
> Sentence Generated:
> [generation results]
> ###

To achieve better performance than 0-shot prompting, we used a small portion of the training set to fine-tune the GPT-3 Curie model, and its results are reasonable:

> Generate a sentence using the following words:
> Pete, transform, Fifth, into
> ###
> Sentence Generated:
> Pete transforms into Fifth.
> ###

3.4 GraphPlan Model

The idea of this method comes from the GraphPlan paper [Chen et al.2021]. It also follow the same paradigm as the previous event-to-event LSTM model: first extract event(s) from sentence(s); then predict the next event; lastly, turn the event(s) into a natural sentence.

3.4.1 Event Extraction The first step is still vectorize each sentence into one or more events. We adapts the same event extraction procedure described in Section 3.3.1, where the events in this model are also represented in the form of $< Subject, Verb, Object, Modifier >$.

3.3.2 Next Event Generation With the event extractions, the main idea of GraphPlan model is to initiate a beam search in a pre-extracted events library based on the coherence score between previous event and potential events. In this way, the model would pick the most likely and most coherent next-event of every predecessor event. In practice, we adopt Bert encoder as the base encoder of event elements (i.e. subject, verb, etc.) and simply concatenate their Bert embedding as the representation for a given event. The event vector is sent to the compositional model in which each two events' embeddings are further concatenate together as the event-pair representation. The last step is to train a classifier model with contrastive learning technique to enables it distinguish valid (i.e. coherent and relevant) event pairs.

In other words, this method accomplish the exact same job of next-event generation described in Section 3.3.2 in a different, more sample-focus fashion.

3.4.3 Event to Natural Language Generation In the last step, we also adapt the same event-to-sentence model from the previous model. See Section 3.3.3 for more details.

4. Question Answering about Game State

The second part of TurtleSoup is the question answering module, which handles player's question about the story, gives yes-or-no answer and keeps track of player's progress. We consider it as a traditional task of question-answering with context. To make it easier to track the state of the player (i.e. which parts of the story the player guessed correctly), we consider the story as a sequence of sentence $[s_1, s_2, ..., s_n]$. Given such a story and player's question s_p, we propose to train or fine-tune three sets of models on our manually annotated data, $([s_1, s_2, ..., s_n], s_p)$ pairs, to decide if the player's question is entailed by the story.

4.1 Few-shot Learning and Fine-tuning on GPT-3

The first method is GPT-3 model for story question answering. For few-shot learning, three randomly sampled story-and-question pairs with answers serve as the few-shot prompt, followed by a new story and question pair for prediction. The output of GPT-3 output is parsed into "yes" or "no". For fine-tuning, the story and question pairs are used as the prompt and "yes" or "no" as the completion.

4.2 BERT for Sequence Classification

Despite the setting of question answering, the nature of binary output of the system makes it a classification problem. Based on [Clark et al.2019]'s experimental results, we first fine-tuned pretrained BERT directly on our yes-no dataset. The results are not satisfactory because our dataset is too small. Therefore, we choose to fine-tune on the large entailment dataset MultiNLI [Williams, Nangia, and Bowman2018] before fine-tuning on our dataset to make the transfer.

4.3 Recurrent Neural Networks

Besides pre-trained models, recurrent neural networks are also tested. We used the Enhanced Sequential Inference Model (ESIM) [Chen et al.2017] architecture. Training first on MultiNLI dataset, we then transfer it by fine-tuning on our Puzzle dataset.

5. Dataset

There are three parts of our work that require a story corpus dataset: the training part of the event-to-event generation model (both the event-to-event LSTM and the GraphPlan model), the evaluation phase of the entire coherence story generation module, and the fine-tuning and evaluation of the question-answering module.

In the first part, a complete and clean story corpus is required to fulfill the coherence assumption between consecutive events (i.e. sentences). In the second one, only appropriate opening sentences are required to mimic the most likely starting stories our system would possibly get in real-time deployment. In the third one, relevant quesion-story pairs are required. Based on these requirements, the following datasets are included in our study.

5.1 Scifi-TV-Shows Dataset

The Scifi-TV-Shows dataset is used in event-to-event model training, discussed in event-to-event LSTM (Secion 3.3.2) and GraphPlan model (Section 3.4.2). It consists of 2276 stories from science fiction TV show plot synopses from Fandom.com wikis [Ammanabrolu et al.2020] (each episode is considered as a story in this dataset). This dataset is pre-processed to contain both the original sentences (key: $sent$) and the corresponding, extracted events (key: $event$ and $event\_gen$ for original text event and generalized event) using the same method discussed in Section 3.3.1 (i.e. $< Subject, Verb, Object, Modifier >$). See examples and table below.

Here are some statstics about the dataset:

	Train	Val.	Test
# Sentences	257108	32855	30938
# Stories	1737	194	450
Avg. # Events	1.5940	1.5681	1.7024

Table 3: Dataset Overview

Here are some examples from the dataset.

- Story #1
  - $sent$: They perhaps need that ship in the future to fend off the.
  - $event$: [they, need, ship, future]
  - $event\_gen$: [(PRP), (want-32.1), (craft.n.02), (DATE)0, in]
- Story #2
  - $sent$: He gets angry about the situation.
  - $event$: [he, get, angry, EmptyParameter]
  - $event\_gen$:[(PRP), (escape-51.1), (angry.a.01), ∅]

5.2 Situation Puzzle Dataset

Another dataset we used is a set of Situation Puzzles from the personal website of Jed Hartman [Hartman2022]. We selected 30 appropriate story beginnings, which are one or more sentences of texts, as the beginning sentence(s) to test TurtleSoup. Among those beginnings, 20 of them serve as the test set and the other 10 are the development set. The test set are not seen by the members who are evaluating the stories while the story beginnings from the development set are used when building and validating the TurtleSoup.

In order to pick the appropriate story beginnings that would not make it too hard for the GPT-3 model to do the generation, we make sure that the story beginnings are more than five words in length and does not contain any ambiguous phrases. For instance, stories that emphasize polysemy, double meanings, or homophones are not included in either test or development set.

We also evaluated the question-answering model on the Kith dataset, where the ground truths are done by our manual annotation. The split of the Kith dataset was into 230 training samples and 52 testing samples.

6. Experiment and Evaluation

We plan to test our story generation module with manual qualitative evaluation; for the question-answering model, on the other hand, we are going to evaluate the accuracy on the held-out annotation from ourselves.

6.1 Story Generation

6.1.1 Generated Stories We tried 5 models for story generations in total, but not all of them succeed in generating meaningful story given an input sentence. In particular, the GraphPlan model and Event-to-event LSTM model both suffer from collapsing issue: they both end up generating a set of unchanged output, which are both highly common events from the training corpus. Therefore, we did not include those two models in our evaluation. We only include the following three models:

- Vanilla-GPT: the baseline model discussed in Section 3.1
- Twisting-GPT: the controlled story generation model discussed in Section 3.2
- Twisting-GPT with Preset: same setting as the previous model, just with additional story preset information discussed in Section 3.2.1

6.1.2 Story Evaluation Metrics Story scoring is broken down into the following three metrics, each with score from 1 to 5. In the end, all the scores for every metric (from all the annotators) are averaged get an overall metric score for each model (see Table 4).

- Coherence: whether the story is logically coherent; if there is repetition in the story content, coherence score should be low;
- Interestingness: the content richness of the story;
- Surprisingness: the twisting effect in the story; if there are unexpected developments in the story, surprisingness score should be low;

6.1.3 Evaluation Results As shown in the Table 4, the two GPT-3 models with twisting effects introduced significantly higher surprisingness to the generated stories.

The introduction of story preset empirically improves the interestingness of the story, which measures the content richness of generated stories. But the difference is not significantly large.

In terms of coherence, the two twisting-GPT models suffer from bad coherence due to the two-step procedure during generation. This would be a potential problem to be addressed in the future work.

	Vanilla	Twisting	Twisting w/ Preset
Coherence	**3.69**	2.64	2.73
Interestingness	2.36	2.92	**3.07**
Surprisingness	2.43	**3.17**	**3.21**

Table 4: Story Generation Evaluations

6.2 Question Answering

For question answering, we evaluate the model on manually annotated questions on the Kith dataset, the Puzzle dataset. We split the Puzzle dataset into 230 training samples and 52 testing samples. The results on the test set are summarized in Table 5.

We parse the generative output of GPT-3 models as follows. For few-shot GPT-3, the output is always a single ”yes” or ”no” string, which can be directly used. For fine-tuned GPT-3, the output is often a sequence of words instead of simple ”yes” or ”no” strings. However, the sequence all starts with ”yes” or ”no”. We therefore consider the start of the sequence as the actual prediction and compare it with the ground-truth.

Model	Data	Accuracy
Majority	-	0.538
GPT-3 Few-shot	-	0.857
GPT-3 Fine-tuning	Puzzle	0.673
BERT	Puzzle	0.554
	MultiNLI + Puzzle	0.681
Recurrent (ESIM)	MultiNLI + Puzzle	0.673

Table 5: Question Answering Results

7. Analysis and Discussions

7.1 Question Answering

The GPT-3 Few-shot model out-performs all other models by a large margin. It is probably because answering the questions requires a high level understanding of the stories. GPT-3, through pretraining, has a strong ability of reading comprehension. Fine-tuning on GPT-3 fails largely due to the limited size of our Puzzle dataset. BERT fine-tuned on MultiNLI and then Puzzle has the second highest accuracy because MultiNLI can be transfered effectively to the task of yes/no question-answering. It is worth noting that ESIM, with orders of magnitude less parameters, achieved comparable results to BERT.

7.2 Effective Story Generation with Controlled Prompts

During our generation, we adopted a step-by-step method, generating the prompt for next generation depending on the previous outcomes. The structured, recurrent GPT-3 pipeline we devised relied heavily upon up-to-time estimation, both of the situation present at the start of the story (prompt) and of the current situation at the particular round. Thus, a decent extraction and manipulation of these meta-information proved crucial to the performance of our system.

The story preset extraction from the first sentence was important for the model to generate more thoughtful, vivid scenarios, instead of repetitively generating the same sentence. For example, in one of the stories, our model predicted successfully that, if something happened during a person's way to work, his boss / company / colleagues might be present nearby and help out, while in the ordinary GPT-3 model, this was never mentioned.

The sentiment module and choice of randomized, proper slots for twisting are also highly effective. With the system in hand, we avoid GPT-3 failures where it refuses to generate anything new. Our model thus is able to control the flow of the story, while also based on a measured amount of impact to maintain coherence. The balance between plot twists and logical coherence is one of the best characteristics of our solution.

7.3 Bias in GPT-3 Generation Results

Obvious bias in large language model is a both interesting and tough part during development. For instance, GPT-3 tends to tell a dead people story when given extremely sad prompt, as well as stating there is always a people winning lottery when it received the precondition that something extremely happy happened. This also happens in its inference phrase, even setting it with high randomness, the answer it inference is the most straightforward. We induce that to the tremendous corpus pretraining, which limits the GPT-3 to the most common events and expansions. While this common sense indeed provides hard support for story logicality and reality, it alleviates our chance to generate unexpected and creative plots.

One efficient way to solve this problem is to integrate structural models and complex planning of events. By weaving these small and straightforward events into a larger causal graph, the hard bias and homogeneity could be mitigated in a more diverse and complex story. We also argue that the bias in large language models is an important factor in generation task limiting the creativeness and fairness of its generation quality, and it is crucial and worthwhile to explore and optimize.

7.4 Unexpected Obstacles in Project Management

In terms of project planning and management, we encountered unexpected obstacles and learned a harsh lesson. Although all of the members are passionate and hardworking, what we planned at the very beginning was far from what we were eventually capable of finishing. Looking back, the key point that we lost most of our time and got ourselves behind the schedule is the delay caused by event-to-event model training. In fact, for a course project like this, it is extremely challenging to train a new end-to-end model, even with referencing papers and code. Instead, per-trained models are much more appropriate.

In addition, we were overly ambitious to build both question-answering and story-generation module in a single project. The problem is not about the ambition, but about the fact that we did not implement the minimum-viable-prototype to validate our idea before working on the whole pipeline. If we tried to implement a minimum-viable-prototype, such as an end-to-end event prediction model with completed event extraction and next-event generator, we could have realize the challenge and turn to the GPT-3 model sooner. However, what ended up happening is that we stuck on the next-event generator, which is a tiny part of the whole system, without realizing the complicate nature of building an end-to-end model.

Luckily, the prompt-guided GPT-3 generation model works well. But in fact, it was built within 6 days of submitting this report (i.e. started 4 days before report due). Next time, we would plan and work on projects more wisely.

8. Future Works

8.1 Story State Tracker

In the TurtleSoup, each newly-generated story segment goes through a verification module before adding to the end of existing story. For now the verification module is simply a sentence level similarity check: whether or not one or more of these new sentences are simply repeating what has been mentioned beforehand. However, as discussed in the Analysis & Discussion section, there are still some repetition in the final generated stories, even when we re-generate with different transitional prompts.

Therefore, what that could significantly increase the robustness of the story generation module is to implement a story state tracker that has the following two functionalities: first, it can verify the actual content of the newly generated sentences not only from the semantic perspective, but also in terms of logical coherence to the previous story state, for example, if a woman is dead, she probably should not die again in the later story; second, it can provide information to build better transitional prompts by guiding the GPT-3 model to generate new information related to other characters or other settings in the story. One way of potentially implement such a story state tracker is to build and maintain a schema tracker with NER extraction tools such as Stanford CoreNLP and commonsense knowledge graph such as ATOMIC.

In addition, current question-answering module only focus on the generated natural language text. With the story state tracker, the question-answering module would also be able to access the ”state” of the story as it develops. In this way, it would also improve the question-answering performance.

8.2 Planning & Script

Surprising but logically coherent is the most important characteristics of situation puzzle stories. In order to fully implement a strict logical chain and complex story structure at the same time, planning with story script is also a potentially helpful in addition to the story state tracker module to add to TurtleSoup. Combining structural model like GNN and introducing causality of events to build a planning module is another essential future target. With a planning module, some event representation would be generated before the story generation, given pre-defined, randomized story preset such as genre, story length, degree of twisting, etc. In this way, the story would actually intriguing since it would follow an actual, hidden story-line instead of following whatever the GPT-3 model considers ”likely”, which is actually only from the language perspective, not from the storytelling perspective.

8.3 Prompt Tuning

Another potential future work would be better prompts in our generation module. For now the TurtleSoup is using explicit text prompt templates for transitional effects, for example, ”Then, something {adverb} {sentiment} happened”. One potential improvement to this prompt building paradigm is to developing robust prompts representation of situation puzzles task, which is more generalized and has larger potentials to introduce better contents than text templates. What's more, tuned prompt representation could acquire the prior knowledge contained in the human-written situation puzzles and has prospective in mining some latent structures in twisting and reversal settings. For instance, a prompt representation about whether it is essential to reverse in current situation and how strong it should be would be needed so that the puzzles could be more natural and intriguing, which is hard to achieve only by text templates.

9. Summary

In short, we tried various ways to build a GPT-3 based system that can accomplish both story generation and questions about the story content. In particular, traditional sequence to sequence models suffer from challenging training setup while the extracted story presets and the specially designed prompts with GPT-3 few-shot completion setup achieves decent story generation results. In terms of the question-answering part, it also benefits greatly from GPT-3 model due to its ability of context understanding. In conclusion, the GPT-3 model is indeed an effective foundation for an automatic situation puzzle system.

References

[Ammanabrolu et al.2020] Ammanabrolu, P.; Tien, E.; Cheung, W.; Luo, Z.; Ma, W.; Martin, L. J.; and Riedl, M. O. 2020. Story realization: Expanding plot events into sentences. volume 34.

[Brown et al.2020] Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; and et al. 2020. Language models are few-shot learners.

[Chen et al.2017] Chen, Q.; Zhu, X.; Ling, Z.-H.; Wei, S.; Jiang, H.; and Inkpen, D. 2017. Enhanced LSTM for natural language inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1657–1668. Vancouver, Canada: Association for Computational Linguistics.

[Chen et al.2021] Chen, H.; Shu, R.; Takamura, H.; and Nakayama, H. 2021. Graphplan: Story generation by planning with event graph. *arXiv preprint arXiv:2102.02977*.

[Clark et al.2019] Clark, C.; Lee, K.; Chang, M.-W.; Kwiatkowski, T.; Collins, M.; and Toutanova, K. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

[Finkel, Grenager, and Manning2005] Finkel, J. R.; Grenager, T.; and Manning, C. D. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting of the association for computational linguistics (ACL'05)*, 363–370.

[Hartman2022] Hartman, J. 2022. Jed's list of situation puzzles.

[hug] Huggingface: Bert for sequence classification. `https://huggingface.co/docs/transformers/model_doc/bert#transformers.BertForSequenceClassification`. Accessed: 2022-04-05.

[Jelodar et al.2019] Jelodar, H.; Wang, Y.; Yuan, C.; Feng, X.; Jiang, X.; Li, Y.; and Zhao, L. 2019. Latent dirichlet allocation (lda) and topic modeling: models, applications, a survey. *Multimedia Tools and Applications* 78(11):15169–15211.

[Liu et al.2021] Liu, P.; Yuan, W.; Fu, J.; Jiang, Z.; Hayashi, H.; and Neubig, G. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing.

[Martin et al.2018] Martin, L.; Ammanabrolu, P.; Wang, X.; Hancock, W.; Singh, S.; Harrison, B.; and Riedl, M. 2018. Event representations for automated story generation with deep neural nets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[Miller1995] Miller, G. A. 1995. Wordnet: a lexical database for english. *Communications of the ACM* 38(11):39–41.

[Reimers and Gurevych2019] Reimers, N., and Gurevych, I. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

[Schuler2005] Schuler, K. K. 2005. *VerbNet: A broad-coverage, comprehensive verb lexicon*. University of Pennsylvania.

[Trinh and Le2019] Trinh, T. H., and Le, Q. V. 2019. A simple method for commonsense reasoning.

[Williams, Nangia, and Bowman2018] Williams, A.; Nangia, N.; and Bowman, S. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 1112–1122. New Orleans, Louisiana: Association for Computational Linguistics.

[Yang et al.2021] Yang, Y.; Pan, B.; Cai, D.; and Sun, H. 2021. Topnet: Learning from neural topic model to generate long stories. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 1997–2005.

[Yao et al.2019] Yao, L.; Peng, N.; Weischedel, R.; Knight, K.; Zhao, D.; and Yan, R. 2019. Plan-and-write: Towards better automatic storytelling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7378–7385.

Appendix

Story Generation Example #1

- (**Story Beginning Sentence**) The snake slithered across the deck, its long body leaving a trail behind it.
- (**Baseline GPT-3**) The snake slithered across the deck, its long body leaving a trail behind it. The snake slithered across the deck, its long body leaving a trail behind it. The passengers on the boat watched in horror as the snake made its way towards them. Suddenly, the snake struck, biting one of the passengers on the leg. The victim screamed in pain as the venom coursed through his

body. The other passengers scrambled to get away from the snake, but it was too late. The snake had already injected its venom into several of them. Within minutes, the passengers were all lying on the deck, their bodies writhing in pain. The boat continued to move down the river, carrying its deadly cargo with it.

- (**Twisting GPT-3**) The boat was moving along a river in india when a large snake dropped onto the deck. The boat's passengers were terrified when a large snake suddenly dropped onto the deck. the snake slithered around for a bit before disappearing back into the water. It's not clear what kind of snake it was, but it was definitely large and scary-looking. the passengers were probably relieved when it disappeared back into the water. There are many different types of snakes in india, so it is difficult to say which one it was without more information. some of the more common species include the indian cobra, russell's viper, and the common krait. these snakes can range in size from a few feet to over 10 feet long, so the snake on the boat was definitely large. The boat's passengers were suddenly attacked by a swarm of bees. the bees were attracted to the boat by the snake, and they began stinging the passengers. the passengers were in pain and some of them began to panic. The snake suddenly reappeared and began eating the bees. the passengers were surprised and relieved to see that the snake was helping them. the snake ate many of the bees and then disappeared again. The boat suddenly capsized and everyone fell into the water. the passengers were terrified and began to panic. some of them tried to swim to shore, but the current was too strong. The boat capsized because of the weight of the snake and the bees. the passengers were terrified and some of them drowned. it is not clear what happened to the snake.
- (**Twisting GPT-3 with story presets**) The boat was moving along a river in india when a large snake dropped onto the deck. The boatman started shouting and the other passengers on the boat began to panic. the main character tried to calm everyone down, but it was difficult to do with the snake hissing and coiling itself around the boat. then, suddenly, the snake spoke. The snake spoke in a human voice, asking why everyone was so afraid. the main character explained that they thought the snake was going to attack them. the snake said that it was just looking for a place to rest and wasn't going to hurt anyone. After the snake had calmed everyone down, it slithered off the boat and into the water. the main character watched it swim away, wondering if they had really just talked to a snake. The boat began to move faster, as if it had a mind of its own. the boatman tried to steer it, but it was no use. the boat was headed for a waterfall and everyone on board knew they were going to die.

Story Generation Example #2

- (**Story Beginning Sentence**) He is a passenger in an airplane and sees the bird get sucked into an engine at 20,000 feet.
- (**Baseline GPT-3**) He is a passenger in an airplane and sees the bird get sucked into an engine at 20,000 feet. The bird got sucked into the engine at 20,000 feet and the engine exploded. The airplane started to lose altitude and the passengers started to panic. The pilot was able to keep the airplane from crashing, but it was a close call.
- (**Twisting GPT-3**) He is a passenger in an airplane and sees the bird get sucked into an engine at 20,000 feet. The bird is most likely dead. The bird's body would most likely be ripped apart by the engine's blades. The bird's body was not ripped apart by the engine's blades and it survived the incident. The bird's body was not ripped apart by the engine's blades and it survived the incident. The bird's body was not ripped apart by the engine's blades and it survived the incident. thus, something happened. The bird's body was not ripped apart by the engine's blades and it survived the incident. this means that the bird was either very lucky or that the engine did not have enough power to rip the bird's body apart. The bird's body was not ripped apart by the engine's blades and it survived the incident. this means that the bird was either very lucky or that the engine did not have enough power to rip the bird's body apart.
- (**Twisting GPT-3 with story presets**) He is a passenger in an airplane and sees the bird get sucked into an engine at 20,000 feet. The airplane began to lose altitude rapidly and the passengers started to panic. the pilot came on the intercom and said that they were losing one of their engines and needed to make an emergency landing. the plane landed safely, but the experience was definitely a supernatural one for the main character. When the plane landed, the main character saw that the bird had somehow made its way into the cabin. it was flapping around and causing a panic. the flight attendants tried to catch it, but it was too fast. The bird transformed into a human being right before everyone's eyes. the main character was shocked, but the other passengers and flight attendants didn't seem to notice. the human-bird hybrid started to fly around the cabin and everyone started to clap and cheer. The human-bird hybrid started to speak in a language that no one could understand. the main character was the only one who could understand what it was saying. it said that it was from a place called "the other side" and that it was here to help them. The human-bird hybrid then went to each person in the cabin and gave them a message from the other side. the main character was the last one to receive a message. the human-bird hybrid told him that he was going to die soon, but that it was not his time yet.